# Conception, réalisation et évaluation d'un jeu sérieux de stratégie temps réel pour l'apprentissage des fondamentaux de la programmation

Mathieu Muratet

▶ **To cite this version:**

Mathieu Muratet. Conception, réalisation et évaluation d'un jeu sérieux de stratégie temps réel pour l'apprentissage des fondamentaux de la programmation. Informatique [cs]. Université Paul Sabatier - Toulouse III, 2010. Français. NNT : . tel-00554287

**HAL Id: tel-00554287**

**https://theses.hal.science/tel-00554287**

Submitted on 10 Jan 2011

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.


# THÈSE

**En vue de l'obtention du**

## DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

**Délivré par** *l'Université Toulouse III - Paul Sabatier*
**Discipline ou spécialité :** *Informatique*

---

**Présentée et soutenue par** *Mathieu MURATET*
**Le** *2 décembre 2010*

**Titre :** *Conception, réalisation et évaluation d'un jeu sérieux de stratégie temps réel pour l'apprentissage des fondamentaux de la programmation*

---

**JURY**

| | |
|---|---|
| *Rapporteurs :* | *Pr. Jean-Jacques BOURDIN (Président du jury)* |
| | *Pr. Pascal ESTRAILLIER* |
| *Examinatrices :* | *Mme Elisabeth DELOZANNE* |
| | *Mme Fabienne VIALLET* |
| *Directeur de Thèse :* | *Pr. Jean-Pierre JESSEL* |
| *Encadrant :* | *M. Patrice TORGUET* |

---

**Ecole doctorale :** *Mathématiques, Informatique et Télécommunication de Toulouse*
**Unité de recherche :** *Institut de Recherche en Informatique de Toulouse*
**Directeur de Thèse :** *Pr. Jean-Pierre JESSEL*

**AUTEUR :** Mathieu MURATET
**TITRE :** Conception, réalisation et évaluation d'un jeu sérieux de stratégie temps réel pour l'apprentissage des fondamentaux de la programmation
**DIRECTEUR DE THÈSE :** Pr. Jean-Pierre JESSEL
**LIEU ET DATE DE SOUTENANCE :** 2 décembre 2010 à l'IRIT-UPS (Toulouse)

---

**RÉSUMÉ :**

Les jeux vidéo font aujourd'hui partie de la culture de nombreux étudiants au même titre que la télévision, les films ou les livres. Or depuis quelques années, les étudiants se détournent des sciences. La recherche dans le domaine de l'enseignement de l'informatique aborde les problèmes du recrutement et du maintien des étudiants dans les formations informatiques. Une approche prometteuse consiste à utiliser la culture vidéoludique des étudiants pour les motiver à investir du temps dans la pratique de la programmation.
Dans ce cadre, les travaux présentés portent sur la conception, la réalisation et l'évaluation d'un jeu sérieux pour l'apprentissage des fondamentaux de la programmation. Ce jeu est basé sur un jeu de stratégie temps réel où la programmation est un moyen d'interaction. Grâce au système Prog&Play, le jeu sérieux a pu être déployé et évalué dans différents contextes d'enseignements.

---

**MOTS-CLÉS :** Jeux sérieux, Apprentissage de la programmation, Prog&Play, Jeux de stratégie temps réel.

---

**DISCIPLINE ADMINISTRATIVE :** Informatique

---

**LABORATOIRE :** Institut de Recherche en Informatique de Toulouse (IRIT)
Université Paul Sabatier, Toulouse III
118 route de Narbonne
31062 Toulouse Cedex 9


# THÈSE

**En vue de l'obtention du**

## DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

**Délivré par** *l'Université Toulouse III - Paul Sabatier*
**Discipline ou spécialité :** *Informatique*

---

**Présentée et soutenue par** *Mathieu MURATET*
**Le** *2 décembre 2010*

**Titre :** *Conception, réalisation et évaluation d'un jeu sérieux de stratégie temps réel pour l'apprentissage des fondamentaux de la programmation*

---

**JURY**

| | |
|---|---|
| *Rapporteurs :* | *Pr. Jean-Jacques BOURDIN (Président du jury)* |
| | *Pr. Pascal ESTRAILLIER* |
| *Examinatrices :* | *Mme Elisabeth DELOZANNE* |
| | *Mme Fabienne VIALLET* |
| *Directeur de Thèse :* | *Pr. Jean-Pierre JESSEL* |
| *Encadrant :* | *M. Patrice TORGUET* |

---

**Ecole doctorale :** *Mathématiques, Informatique et Télécommunication de Toulouse*
**Unité de recherche :** *Institut de Recherche en Informatique de Toulouse*
**Directeur de Thèse :** *Pr. Jean-Pierre JESSEL*

# Remerciements

Je tiens à remercier les membres du jury pour avoir accepté d'évaluer mes travaux ainsi que pour leurs commentaires avisés. Je remercie donc en tout premier lieu mes rapporteurs, les professeurs Jean-Jacques Bourdin et Pascal Estraillier, ainsi que les examinatrices, Elisabeth Delozanne et Fabienne Viallet. À toutes les deux, je tiens à exprimer ma profonde reconnaissance. Elisabeth Delozanne et sa collègue Françoise Le Calvez ont, par leur persévérance et leur implication, participé au déploiement du jeu sérieux à l'extérieur de l'Université Paul Sabatier. Fabienne Viallet par son investissement personnel m'a aidé à révéler la composante pédagogique de mes travaux. Cette collaboration avec Fabienne Viallet a été, pour moi, l'une des rencontres les plus enrichissantes lors de ces années de recherche.

La réalisation de cette thèse est l'aboutissement de mes études universitaires. Le point de départ de cette aventure commence en première année de Master où Patrice Torguet accepte de m'encadrer pour un Travail d'Étude et de Recherche. Il accepte de renouveler l'expérience l'année suivante pour mon stage de Master Recherche. La réalisation de cette thèse en est la suite logique. Pour sa confiance, sa disponibilité et son encadrement tout au long de ces années, je le remercie avec la plus profonde sincérité. Je remercie également Jean-Pierre Jessel pour m'avoir accueilli dans l'équipe VORTEX et fourni un cadre de travail idéal pour la réalisation de cette thèse. Enfin, je remercie Monsieur Luis Fariñas del Cerro, directeur de l'IRIT, pour m'avoir accueilli dans son laboratoire.

Je tiens également à préciser que ce travail n'aurait jamais pu être réalisé sans la collaboration de nombreux enseignants. Je tiens donc sincèrement à remercier l'ensemble des équipes pédagogiques qui m'ont fait confiance. Je remercie donc Max Chevalier et Christian Percebois (département Informatique - site Toulousain), Jérémie Guiochet et André Lozes (département Génie Électrique et Informatique Industrielle - site Toulousain) et Sylvain Barreau et Emmanuel Conchon (département Services et Réseaux de Communication - site Castrais) de l'IUT A de Toulouse, mais aussi Marie-Françoise Canu et André Péninou du département Informatique de l'IUT B de Toulouse et enfin Véronique Gaildrat, Mathias Paulin et tous les enseignants de l'Université Paul Sabatier qui ont participé aux expérimentations. Sans eux, je n'aurais jamais pu tester le jeu dans des contextes d'enseignements réels et ainsi mettre en place les évaluations… merci encore.

Mes collègues et amis rencontrés au cours de ces quelques années m'ont également apportés de nombreux moment de détente. Je remercie donc ceux qui m'ont accompagné tout au long de

cette thèse, merci à Jonathan Claustre, Andra Doran, Marion Dunyach, Philippe Ercolessi, Guillaume Gales, Mathieu Giorgino, Dorian Gomez, Olivier Gourmel, François Lefebvre-Albaret, Sylvain Lemouzy, Anthony Pajot, Marie Ploquin, Samir Torki et Rodolphe Vaillant-David. Je remercie également ceux qui m'ont précédé et qui m'ont inspiré, merci à Souad El Merhebi, Vincent Forest et Jean-Christophe Hoelt. Pour tout ce temps passé en leur compagnie, pour leur aide, pour les « grandes » discussions au restaurant universitaire, pour les mémorables parties de Magic et les folles courses de karting, je leur dis à tous un grand merci. Je tiens enfin à remercier Loïc Barthe, Nelly de Bonnefoy et Roger Pujado pour leur sympathie et leur bonne humeur.

Enfin je souhaite remercier ma famille pour son soutien et le temps passé à la relecture de mes publications francophones. Je remercie tout particulièrement mes parents Joël et Brigitte Muratet. Toutefois, ma plus sincère reconnaissance est pour mon épouse, Céline. Elle a su m'épauler à chaque instant et me donner la motivation suffisante pour mener à bien ce projet. Je lui dédie cette thèse.

# Table des matières

# Table des figures

# Introduction

Depuis la première explosion des jeux vidéo dans les années 80, l'industrie du jeu vidéo a pris une place importante dans l'économie mondiale. En 2008 le marché du jeu vidéo aux États-Unis a atteint 11,5 milliards de dollars [Ass09] et a dépassé le marché du cinéma (10 milliards de dollars en 2008) [1]. Les étudiants, actuellement à l'université, ont grandi avec les jeux vidéo. Cette composante ludique fait partie de leur culture au même titre que la télévision, les films ou les livres.

Par ailleurs, les étudiants se détournent des disciplines scientifiques. En informatique, par exemple, de nombreux travaux attestent que le nombre d'étudiants est en chute libre et que régulièrement 50% ou plus des étudiants ayant initialement choisi des études en informatique décident rapidement d'abandonner. L'Université Paul Sabatier subit le même phénomène avec une baisse de 17,6% d'étudiants inscrits en première année de licence sur les quatre dernières années et une baisse de 30% en deuxième année de licence informatique sur les sept dernières années.

La recherche dans le domaine de l'enseignement de l'informatique consacre une part importante de ses travaux aux problèmes du recrutement et du maintien des étudiants dans les formations informatiques. Une approche prometteuse consiste à utiliser la culture vidéoludique des étudiants pour les motiver à investir du temps dans la pratique de la programmation.

Forte de ces constats, la recherche présentée dans ce mémoire s'attache à étudier un jeu sérieux pour l'apprentissage de la programmation, le recrutement et le maintien des étudiants dans la filière informatique. Ce travail de recherche est centré sur un type de jeu particulier : les jeux de stratégie temps réel (STR). Comme nous le montrerons dans la suite, ce type de jeu est bien adapté à la mise en œuvre d'exercices de programmation et constitue donc un cadre de conception adapté à la problématique.

Ce travail initié dans le domaine de l'informatique a nécessité des compétences en didactique pour aborder les problèmes théoriques liés aux contenus des formations en informatique et à l'apprentissage de la programmation. Dès la deuxième année de mes recherches, j'ai collaboré avec le Service Universitaire de Pédagogie (SUP) de l'Université Paul Sabatier qui, à son tour, m'a orienté vers Fabienne Viallet, membre de l'Unité Mixte de Recherche « Education, Formation, Travail, Savoirs » (UMR EFTS). Ce partenariat m'a également permis de concevoir le cadre d'évaluation du projet pour les différentes expérimentations.

Dans ce contexte, ce document traite des problématiques liées à la conception et à la réalisation d'un jeu sérieux centré sur les fondamentaux de la programmation et de son évaluation

1. `http://www.the-numbers.com/market/` accédé le 27 Août 2010.

en contexte réel. La présentation des travaux s'articule donc autour de deux parties principales. La première s'attache à réaliser une étude sur les jeux sérieux et l'apprentissage de la programmation. La seconde partie traite des contributions réalisées au cours de ces travaux.

L'état de l'art est structuré en trois chapitres qui présentent de manière détaillée les composantes relatives au cadre de recherche.

Le premier chapitre s'attache à positionner les jeux sérieux vis-à-vis des jeux vidéo et des simulateurs. En effet, les frontières entre ces trois types d'applications sont parfois mal définies. L'objectif de cette première étude consiste donc à comprendre les tenants et les aboutissants des jeux sérieux en vue de poser un regard critique sur l'existant.

Le cadre de recherche étant centré sur les jeux de STR, le deuxième chapitre présente ce type de jeux en vue d'en identifier les règles et les buts. Cette présentation s'appuie sur l'étude d'un jeu ayant fortement participé à définir les bases des jeux de STR : *Dune 2*. Suite à cette définition, des exemples de jeux de STR à code source ouvert, susceptibles de servir de base à la conception d'un jeu sérieux, sont examinés.

Le troisième chapitre se consacre à la présentation de l'enseignement en informatique. Comme à notre connaissance, il n'existe pas de rapport détaillé sur l'enseignement de l'informatique en France, l'analyse du modèle anglo-saxon est mis en perspective dans notre vision du modèle français. À cette occasion, ce chapitre aborde le problème de la « *la crise de l'informatique* » dans l'enseignement et présente quelques innovations pédagogiques qui tentent d'y répondre.

Suite à l'analyse de ces innovations pédagogiques, une approche pragmatique est envisagée pour concevoir, réaliser, mettre en œuvre et évaluer un jeu sérieux sur l'apprentissage des fondamentaux de la programmation. Les contributions, réalisées au cours de ces travaux, s'articulent donc autour de la conception du jeu sérieux et de son évaluation.

Le quatrième chapitre présente donc la conception du jeu sérieux qui s'organise autour de deux composantes principales : le développement du système Prog&Play et son intégration aux jeux de stratégie temps réel [MTVJ10a] [MTJV09] [MTJ09] [MTJV08b] [MTJV08a] ; et l'incorporation du savoir à enseigner à travers les différents modes de jeu [MTVJ10a] [MTJV09].

Outre ces deux points clés, ce chapitre est introduit en préambule par une série d'enquêtes dont l'objectif est de vérifier la popularité des jeux de STR auprès d'un échantillon d'étudiants apprenant la programmation [MTJV09] [MTJV08b]. Enfin, ce quatrième chapitre se termine en abordant la problématique de la portabilité du système Prog&Play vers différents langages de

programmation. Cette compatibilité a notamment permis de faciliter la mise en place d'expérimentations dans différents contextes d'enseignements.

Le cinquième chapitre s'attache à présenter ces évaluations à travers la description des contextes d'expérimentation, des protocoles d'évaluation, de l'adaptation des enseignements pour intégrer le jeu et des résultats obtenus [MTVJ10a] [MTVJ10b] [MVTJ09] [VMD+10].

# Première partie

# État de l’art

# 1

# Jeux vidéo, jeux sérieux et simulateurs

Dans le passé, les procédés utilisés dans les simulateurs étaient seulement disponibles pour des systèmes industriels et militaires très onéreux. Avec l'augmentation des performances des ordinateurs, les technologies se sont démocratisées jusqu'à investir des applications du grand public telles que les jeux vidéo. L'utilisation commune de ces technologies par les simulateurs et certains jeux vidéo rend la distinction de ces deux types d'applications parfois floue. Les nouveaux logiciels tels que les jeux sérieux accentuent cette confusion. Nous allons donc présenter ces trois composantes pour tenter de positionner les jeux sérieux vis-à-vis des simulateurs et des jeux vidéo.

## 1.1 Généralités sur les jeux vidéo

Mickael Zyda [Zyd05] définit le jeu vidéo comme « *un défi intellectuel lancé sur un ordinateur selon des règles spécifiques. Il est dédié au divertissement ou au fait de remporter un enjeu* ». Cette définition met en évidence deux points clés qui semblent fondamentaux pour caractériser les jeux vidéo : les **règles** et l'**enjeu**.

Les règles sont une composante fondamentale du jeu en général. Jean Piaget [Pia94] a défini une classification du jeu chez l'enfant :

– le jeu d'exercices (0 à 2 ans) correspond à la période sensorimotrice. Le bébé effectue des exercices moteurs réflexes en fonction des informations perçues ;

– le jeu symbolique (2 à 8 ans) exerce la capacité de l'enfant à représenter une réalité non actuelle, c'est-à-dire l'imitation de l'adulte à travers le jeu (jeu de la poupée, de la dînette, du bricolage) ;
– le jeu de règles (6 à 15 ans) marque la socialisation de l'enfant. Ces jeux sont le reflet du code social (jeux comportant des règles).

Dans le développement de l'enfant, Jean Piaget introduit la règle comme indicateur du dernier stade du jeu. Il précise que « *[...] s'il ne demeure chez l'adulte que quelques résidus des jeux d'exercice simple [...] et des jeux symboliques [...] le jeu de règles subsiste et se développe même durant toute la vie (sports, cartes, échecs, etc.)* » [Pia94, p. 149] ; le jeu vidéo entre donc dans cette dernière catégorie et contribue à promouvoir une communication sociale entre les joueurs par l'échange d'expériences ou de solutions acquises à travers le jeu. Le mode de jeu « multijoueur » favorise d'autant plus cet échange qu'il permet à plusieurs joueurs d'interagir ensemble dans le même environnement virtuel. Alliance, coopération ou compétition émergent pour fournir une forme de communication sociale.

L'enjeu est défini par le dictionnaire TLFi (Trésor de la Langue Française informatisé) comme suit : « *Ce que l'on peut gagner ou perdre dans n'importe quelle entreprise* ». La présence d'enjeu dans les jeux vidéo implique donc la définition de buts afin de déterminer le gain ou la perte en fonction de l'atteinte ou non des objectifs. La stratégie mise en œuvre par le joueur pour atteindre ce but peut être extrêmement dirigée (le joueur ne peut faire que très peu de choix) à l'image du jeu *The House of the Dead*[2] ou, à l'inverse, des jeux comme *Les Sims 2*[3] ou *Grand Theft Auto IV*[4] laissent le joueur presque totalement libre dans ses prises de décisions. L'objectif, quant à lui, n'est pas toujours clairement défini. Par exemple, *Les Sims 2* pointe constamment les difficultés à surmonter par le joueur (fatigue, colère, saleté... voir Figure 1.1). L'approche qui consiste à placer le joueur dans une dynamique de résolution de problèmes successifs est, semble-t-il, une motivation tout aussi grande que d'atteindre un succès clairement défini.

2. Sega, 1998
3. Electronic Arts / Maxis, 2000, `http://thesims.ea.com/` accédé le 26 Avril 2010.
4. Take 2 Interactive / Rockstar, 2008

FIGURE 1.1 – Les Sims 2.
*Le jeu pointe constamment les problèmes à résoudre par le joueur.*

### 1.1.1 Historique

Pour comprendre l'évolution du jeu vidéo, l'historique suivant présente par ordre chronologique quelques dates clés ayant participé à l'avènement du jeu vidéo, tant sur le plan technique que matériel.

- **1952** : *OXO* est le premier jeu graphique fonctionnant sur un ordinateur (l'EDSAC ou *Electronic Delay Storage Automatic Calculator*). Ce jeu de morpion a été créé par A.S. Douglas dans le cadre de sa thèse sur l'interaction homme-ordinateur à l'université de Cambridge.
- **1958** : *Tennis for Two* est un autre précurseur du jeu vidéo créé par William Higinbotham fonctionnant sur oscilloscope et circuit électronique dédié. Il a été conçu pour distraire les visiteurs lors des portes ouvertes du laboratoire national de Brookhaven.
- **1962** : *Space War* a été développé dans le cadre du MIT (*Massachusetts Institute of Technology*) par Steve Russel et d'autres étudiants dans l'objectif de montrer les performances techniques du PDP-1 (*Programmed Data Processor-1*) de la firme DEC (*Digital Equipment Corporation*). Deux joueurs sont nécessaires pour réaliser une partie. Chacun d'eux contrôle un vaisseau spatial soumis à la gravité d'une étoile. L'objectif consiste à tirer sur le vaisseau de l'adversaire tout en contrôlant sa trajectoire pour ne pas entrer en col-

lision avec le soleil. Chaque joueur dispose d'une quantité de carburant et de munitions limités. *Space War* est considéré comme étant le premier jeu vidéo sur ordinateur à avoir inspiré des produits commerciaux. En effet, en septembre 1971, *Galaxy Game* (une version reprogrammée de *Space War* sur un PDP-11) est la première machine de jeu vidéo commerciale (borne d'arcade). Ce jeu conçu en un seul exemplaire a été installé au *Tressider Student Union* de l'université de Stanford. En Novembre de cette même année, *Computer Space* est la première borne d'arcade payante à être distribuée en série. Ce jeu est également inspiré de *Space War*.

- **Septembre 1972** : l'*Odyssey*, distribuée par Magnavox, est la première console de salon. Elle est vendue avec 6 jeux intégrés et un ensemble d'accessoires dont des caches à appliquer sur l'écran de télévision pour simuler le décor du jeu. À la fin de cette même année, Atari développe et commercialise *Pong*. C'est le premier jeu vidéo à remporter un réel succès, il marquera le démarrage de l'industrie du jeu vidéo. *Pong*, *Space Invader* (1978) développé par Taito, *Pac-Man* (1980) développé par Namco et *Tetris* (1984) d'Alexei Pajitnov sont considérés comme de grands classiques de l'histoire des jeux vidéo.
- **1976** : La *Fairchild Channel F* est la première console de jeux vidéo basée sur un système de cartouches. Les consoles des années 80 et début 90 telles que la NES (*Nintendo Entertainment System*) et la super NES de Nintendo ainsi que la *Master System I* et *II* et la *MegaDrive* de Sega reprendront ce système. Il faudra attendre 1988 pour voir apparaître un nouveau média de support de jeux tel que le CD-ROM avec la console *PC Engine* de NEC (*Nippon Electric Company*). Le CD-ROM supplantera le système de cartouches en 1994 avec l'arrivée de la 3D et des consoles cinquième génération (la *Saturn* de Sega et la *Playstation* de Sony).
- **1977** : La console *Atari 2600* contribue à la démocratisation des jeux vidéo en permettant l'émergence de grands classiques tels que *Space Invader* ou *Pac-Man* présentés ci-dessus.
- **1979** : Création de la *Microvision* par MB (*Milton Bradley company*) la première console de jeux vidéo portable équipée d'un écran à cristaux liquides. Le principe de console portable explosera réellement en 1989 avec la sortie de la *Game Boy* de Nintendo.
- **1981** : *Donkey Kong* créé par Nintendo pose les bases du jeu de plateforme et lance l'un des personnages les plus célèbres de l'histoire du jeu vidéo : Mario.
- **1986** : *Habitat* est le premier monde virtuel commercial « en ligne ». Créé par *Lucasfilm Games* en collaboration avec *Quantum Computer Services* (qui est devenu depuis *America Online* (AOL)), cet environnement est représenté par des scènes animées en deux

dimensions dans lesquelles interagissaient des utilisateurs connectés via des modems. *Habitat* tient son origine dans les MUD (*Multiple User Dimensions*), mondes virtuels en mode texte apparus au début des années 80 avec les premiers systèmes « en ligne » (les BBS ou *Bulletin Board Systems*). *Habitat* a réussi à drainer, à l'époque, quinze mille utilisateurs sur une base totale de cent mille utilisateurs de ce type de service « en ligne ». Ces mondes virtuels comme les MUD ou *Habitat*, communément appelés des *ChatWorlds* (littéralement : mondes où l'on bavarde), peuvent être considérés comme les ancêtres des jeux massivement multijoueurs (*Massively Multiplayer Online Game*) ou MMOG modernes comme *Ultima Online* (1997) d'*Origin Systems* et *Electronic Arts*, *Everquest* (1999) de *Verant Interactive* ou *World of Warcraft* (2004) de *Blizzard Entertainment*.

- **1992** : *Wolfenstein 3D* créé par *Id Software* à initié le genre de tir subjectif. Il sera suivi l'année d'après par *Doom* (de *Id Software* également) qui démocratisera ce genre de jeu et le portera au niveau multijoueur. Ce dernier est donc probablement à l'origine de tous les jeux en réseau actuels. Il est communément admis que la version *shareware* de *Doom* a été récupérée quinze millions de fois sur Internet et/ou passée d'individu à individu.
- **1993** : *Dune2* créé par *Westwood Studios* définit un nouveau genre de jeu vidéo en posant les bases des jeux de stratégie temps réel modernes.
- **1998** : Sega lance la sixième génération de consoles avec la *Dreamcast*. Sony et Nintendo ne tardent pas à contre-attaquer en sortant respectivement la *Playstation 2* en 2000 et la *GameCube* en 2001. Cette même année, Sega se retire du marché des consoles de salon avec l'arrêt de la production de la *Dreamcast* et Microsoft lance sa console : la *Xbox*.
- **2006** : Les consoles septième génération voient le jour avec entre autres la *Wii* de Nintendo qui révolutionne la manière de jouer grâce à ces périphériques d'entrée intégrant de nouvelles technologies telles qu'un accéléromètre, le *Bluetooth*, un système de pointage, etc.

Ce bref historique montre que l'évolution du jeu vidéo est fonction des deux composantes matérielle et logicielles. En effet, chaque nouvelle technologie introduite par les différentes générations de consoles et d'ordinateurs a permis la conception de jeux toujours plus innovants. Inversement, ces mêmes jeux qui ont marqués l'histoire par leur popularité ou leur ingéniosité sont des moteurs indispensables au développement des nouvelles technologies.

### 1.1.2 Classification

Le marché du jeu vidéo est donc en augmentation croissante. L'enquête de l'ESA [Ass09] (*Entertainment Software Association* [5]) indique que le nombre de jeux vidéo vendus aux États-Unis est passé de 72,8 millions en 1996 à 298,2 millions en 2008. Cette progression de 300% dénote l'effervescence de cette industrie qui ne cesse de proposer de nouvelles innovations dans l'objectif de toujours surprendre les joueurs. Ce cercle vertueux, sur le plan économique a pour conséquence une augmentation de l'offre vidéo-ludique. Pour mieux appréhender cette diversité, plusieurs tentatives de classification des jeux vidéo ont été proposées.

Chris Crawford [Cra84, chap. 3] présente, dès 1984, sa taxonomie des jeux vidéo. Il insiste sur la nécessité d'un classement afin de révéler les liens et différences entre les familles et les membres d'une même famille de jeux vidéo. Déjà, Crawford est bien conscient de la grande instabilité des jeux vidéo due à l'évolution rapide de ces derniers et n'hésite pas à prédire l'obsolescence ou l'inexactitude de sa propre taxonomie. Crawford définit donc deux grandes catégories de jeux pour représenter l'ensemble des jeux vidéo existants à cette époque : les jeux de dextérité/action et les jeux de stratégie.

Les jeux de dextérité/action font intervenir en priorité la perception et l'habileté motrice. Dans les années 80, cette famille de jeu vidéo est la plus populaire. Elle est caractérisée par des jeux en temps réel qui mettent l'accent sur les graphismes, le son et l'utilisation de joysticks ou de manettes au détriment du clavier. Les compétences premières demandées aux joueurs sont une bonne coordination œil/main et un très faible temps de réaction. Ces types de jeux sont regroupés dans six sous-catégories : les jeux de combats, les jeux de labyrinthes, les jeux de sports, les jeux de « barres » (basés sur *Pong* [6]), les jeux de courses et les jeux divers (ensemble des jeux n'entrant pas complètement dans cette taxonomie).

Les jeux de stratégie mettent plus l'accent sur la réflexion que sur la manipulation. Crawford divise les jeux de stratégie en six catégories : les jeux d'aventure, les jeux de rôle, les jeux de guerre, les jeux de hasard, les jeux éducatifs et les jeux de communication.

Mark J. P. Wolf [Wol02, chap. 6] présente, plus récemment (2002), une classification par genre. Il analyse la classification iconographique du cinéma (drame, policier, comédie, etc.) et étudie la portabilité aux jeux vidéo. Il en déduit qu'une telle classification ne peut être adaptée

5. `http://www.theesa.com/` accédé le 31 Mars 2010.
6. Atari, 1972

au jeu vidéo en raison de la composante interactive qui est, selon lui, un facteur important de l'expérience de jeu. En effet, un même thème comme la science-fiction peut servir de support à différents types de jeux. Dans ce cas, la distinction entre ces jeux porte sur la manière dont le joueur interagit avec le jeu et non sur le contenu du jeu. Wolf définit donc quarante deux genres dont voici quelques exemples :

- *Abstract* : ce type de jeu sans représentation physique implique souvent un objectif non orienté ou organisé par une narration.
- *Capturing* : jeu dont l'objectif principal est la capture d'objets ou de personnages qui s'éloignent et tentent d'esquiver le joueur.
- *Catching* : jeu dont l'objectif principal est la prise d'objets ou de personnages qui n'essaient pas d'éviter activement le joueur.
- *Collecting* : jeu dont l'objectif principal est la collecte d'objets statiques ou en mouvement sur une petite zone.
- *Combat* : jeu qui implique un ou deux joueurs (l'un d'eux pouvant être contrôlé par le jeu) se tirant l'un sur l'autre à l'aide de projectiles. Chaque joueur possède des moyens similaires pour un combat équilibré.
- *Escape* : jeu dont l'objectif principal est l'esquive de poursuivants ou la sortie d'enclos.
- *Fighting* : jeu qui implique le combat de personnages habituellement à main nue, un contre un, sans utilisation d'armes ou de projectiles.
- *Maze* : jeu dont l'objectif consiste à se déplacer avec succès dans un labyrinthe.
- *Shoot 'em up* : jeu dont l'objectif est de tirer sur plusieurs adversaires ou objets en vue généralement de les détruire.
- *Sports* : jeu qui est une adaptation d'un jeu de sport existant ou d'une variation de celui-ci.
- *Strategy* : jeu qui met l'accent sur l'utilisation d'une stratégie. Les actions rapides ou l'utilisation de réflexes ne sont généralement pas nécessaires pour terminer le jeu.
- *Target* : jeu dont l'objectif principal est d'atteindre (ou de tirer) sur des cibles statiques.

Ces quelques exemples illustrent la finesse de cette classification. Ces types de jeux sont identifiés comme parfois très proches les uns des autres à l'image des genres *Shoot 'em up* et *target* ou *catching*, *collecting* et *capturing*. Selon Wolf chaque jeu peut être rattaché à plusieurs genres en raison des différents types d'actions et objectifs présents dans un même jeu.

Julian Alvarez [Alv07, chap. 4, sec. 2.2] (2007), quant à lui, propose une classification basée sur l'étude du *gameplay* des jeux vidéo. Alvarez s'appuie sur la définition de Jean-Noël Portugal

qui définit le *gameplay* comme étant « *les règles du jeu, les buts généraux et locaux attribués au joueur, les moyens d'action et de liberté concédés à l'utilisateur dans l'univers virtuel. Ainsi que les organisations spatiale, temporelle et dramaturgique* ». Après avoir étudié plus de 500 jeux, Alvarez définit 11 briques *gameplay* (*Answer*, *Avoid*, *Block (Maintain)*, *Create*, *Destroy (Collect)*, *Have luck*, *Manage*, *Move*, *Position*, *Shoot* et *Toy*) et 4 métabriques *gameplay* (*Brain* composée de *Answer* et *Avoid*, *God* composée de *Manage* et *Create*, *Killer* composée de *Shoot* et *Destroy (Collect)* et *Driver* composée de *Move* et *Avoid*). Les métabriques sont définies comme « *1 : une combinaison de deux briques gameplay de nature complémentaire qui donne ainsi naissance à un défi. 2 : Ajouter une brique gameplay à une métabrique, confère au défi porté par cette dernière, une variante, qui cependant n'affecte pas sa nature profonde. 3 : Si l'on ajoute plusieurs briques gameplay à une métabrique, le point (2) reste probablement vrai tant que la mise en présence de ces briques gameplay ne constitue pas à leur tour une autre métabrique. 4 : Associer des métabriques revient à associer leur challenge respectif* ».

À partir de ces 4 métabriques, Alvarez définit 15 manières possibles de les combiner. Ces différentes combinaisons correspondent à 15 types de challenges distincts de jeux vidéo. Les briques *gameplay* non utilisées dans une des combinaisons permettent d'identifier les variantes de jeux vidéo pour un type de challenge particulier. Alvarez, par ailleurs, met en évidence une relation inversement proportionnelle entre le nombre de métabriques impliquées dans le challenge et le nombre de variantes possibles de jeu, ceci étant dû au nombre restreint de briques *gameplay* restantes.

La presse spécialisée du jeu vidéo propose également différentes classifications dépendantes des lignes éditoriales de chaque magazine. Ces classifications sont sans nul doute les plus consultées et utilisées par les communautés de joueurs, elles méritent donc d'être mentionnées. Parmi l'ensemble des magazines accessibles en ligne, *GameSpot*[7] est retenu de part sa grande popularité. Selon *Alexa*[8] et *Compete*[9], deux sites web d'analyse du trafic sur Internet, *GameSpot* serait le magazine de jeu vidéo le plus consulté sur Internet en atteignant en moyenne dans le classement global le 220ème rang mondial avec plus de 5000 visiteurs par mois sur l'année 2009. Les jeux répertoriés par ce site sont classés en 36 catégories dont les classiques jeux

7. `http://www.gamespot.com` accédé le 02 Avril 2010
8. `http://www.alexa.com` accédé le 02 Avril 2010
9. `http://siteanalytics.compete.com` accédé le 02 Avril 2010

d'action, d'aventure, de course, de plates-formes, de rôle, de simulation, de sport, de stratégie temps réel et de tir subjectif pour ne citer que ceux-là.

À titre d'exemple, nous présentons le positionnement du jeu *Pac-Man* [10] dans ces diverses classifications. L'objectif de ce jeu consiste à collecter des bonus tout en esquivant les fantômes présents dans le labyrinthe. Pour Crawford, *Pac-Man* est un jeu de dextérité/action appartenant à la sous-catégorie des jeux de labyrinthe. Pour Wolf, ce jeu peut être classé dans les genres *Abstract*, *Collecting*, *Escape* et *Maze*. Pour Alvarez, *Pac-Man* est caractérisé par la métabrique *Driver* et la brique *Destroy (Collect)*. Enfin *GameSpot*, classe *Pac-Man* dans les simples jeux d'action.

Ces quelques exemples de classification illustrent la difficulté d'ordonner les jeux vidéo de manière précise, exhaustive et pérenne. La cause de cette difficulté réside dans une composante importante de la conception d'un jeu vidéo : l'innovation. Celle-ci porte autant sur le logiciel que sur le matériel et a pour objectif de proposer aux joueurs sans cesse de nouvelles sensations. Les classifications sont toutefois nécessaires pour appréhender la masse des jeux vidéo d'une époque donnée. Se pose alors la question du positionnement des « jeux sérieux » vis-à-vis des jeux vidéo.

## 1.2 Généralités sur les jeux sérieux

Le terme « jeu sérieux », tel que nous l'entendons aujourd'hui, semble avoir été évoqué pour la première fois, en 1970, par Clark Abt dans son livre *Serious Games* [Abt70]. Dans cet ouvrage, le terme *Serious Games* est utilisé pour des jeux de cartes ou de plateaux conçus dans un but éducatif.

Actuellement, l'expression « jeu sérieux » est généralement utilisée dans un contexte informatique plus prononcé et caractérise une gamme de jeux vidéo bien plus large que le simple jeu éducatif. En effet, un jeu vidéo conçu avec un objectif autre que le simple divertissement peut souvent être considéré comme un jeu sérieux. Mickael Zyda [Zyd05] reprend sa définition des jeux vidéo pour caractériser le « jeu sérieux » comme « *un défi intellectuel lancé sur un ordinateur selon des règles spécifiques. Il utilise le divertissement pour servir à la formation, à l'éducation, à la santé, à la sécurité civile et comme stratégie de communication dans*

10. Namco, 1980

*des milieux institutionnels ou privés* ». Outre cette définition, Zyda identifie un point critique de la conception d'un jeu sérieux concernant le positionnement du « jeu » par rapport au concept de « sérieux » (message véhiculé qu'il soit d'ordre formatif, éducatif, informatif, etc.). Zyda [Zyd05] écrit à ce propos que « *la pédagogie doit être subordonnée au scénario du jeu – la composante ludique doit primer* ». L'hypothèse considérée est que si un jeu est attractif, amusant, stimulant et encourage le joueur à progresser, alors le joueur intègrera automatiquement les caractéristiques du jeu et de nombreuses informations.

### 1.2.1 Historique et domaines d'application

Très tôt dans l'histoire des civilisations, des activités culturelles sont apparues. Elles servaient à distraire le peuple (jeux olympiques grecs ou jeux du cirque romains). Quelques jeux font également leur apparition comme la « pettie » grecque ou le « latroncule » romain (voir figure 1.2). Ce sont des jeux de prise par encerclement, à connotation guerrière (ancêtres du jeu de dames). Ces jeux n'ont pas pour objectif premier l'éducation ou la résolution de problèmes particuliers. Cependant, ce sont les jeux les plus anciens qui nous sont parvenus, basés sur un raisonnement intellectuel, faisant intervenir la réflexion, la planification et non le hasard.

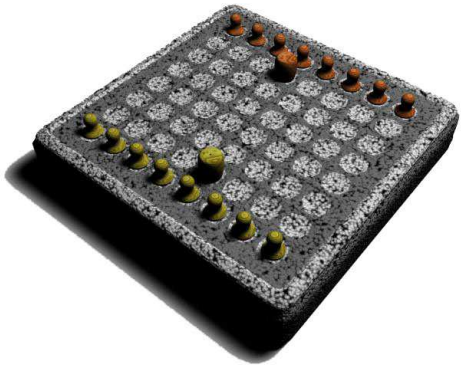


FIGURE 1.2 – Un jeu de « latroncule ».

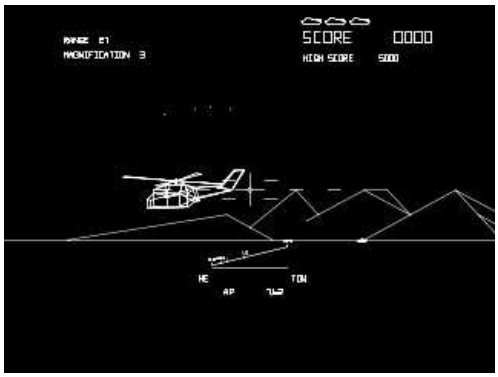



FIGURE 1.3 – Premier jeu sérieux : *Army Battlezone*.

Plus récemment, on pourrait attribuer le rang de premier jeu sérieux à *Army Battlezone* (voir figure 1.3), un projet développé par Atari en 1980, conçu pour l'entraînement des militaires américains.

Par la suite, des groupes variés aux États-Unis et au Royaume-Uni, ont utilisé le principe de l'éducation par le jeu [11] pour évoquer des problèmes sociaux ou de santé tels que la toxicomanie, la vaccination, les grossesses adolescentes, le SIDA et le cancer. En France, les premières tentatives à grande échelle d'éducation par le jeu se retrouvent principalement dans la communauté des musées scientifiques. Le « Palais de la découverte » et la « Cité des sciences et de l'industrie » à Paris sont les premiers exemples à avoir tenté l'expérience. Suite à ces initiatives, un nombre croissant de musées a vu le jour tel que la « Cité de l'espace » à Toulouse, le « Centre National de la Mer Naüsicaa » à Boulogne-sur-Mer, « Vulcania » près de Clermont-Ferrand et « Bioscope » à Ungersheim en Alsace.

La définition de Zyda présentée dans la section 1.2 met en exergue la diversité des domaines d'application dans lesquels les jeux sérieux peuvent être employés. Cette diversité semblant s'accroître continuellement, Julian Alvarez [Alv07, chap. 1, sec. 3] propose six domaines d'application afin de classer les jeux sérieux : militaire, militant, marketing, éducatif/formatif, informatif et médical. Pour illustrer cette hétérogénéité, nous présentons par ordre chronologique quelques exemples de jeux sérieux aux domaines d'application et aux objectifs différents.

FIGURE 1.4 – *America's Army*.

*America's Army* [12] (voir figure 1.4) est un jeu conçu en 2002 par l'institut MOVES (*Modelling, Virtual Environments, and Simulation*) du *Naval Postgraduate School* de Monterey, Californie (Etats-Unis). Il a été initialement développé comme outil de recrutement pour l'armée

11. `http://www.socialimpactgames.com/`
12. `http://www.americasarmy.com/` accédé le 25 Mars 2010.

des Etats-Unis. Il est devenu l'un des premiers jeux sérieux à remporter un réel succès. C'est l'un des dix jeux d'action les plus populaires joué en ligne. Le succès de ce jeu confirme donc le potentiel des jeux sérieux et ouvre la voie vers les autres secteurs d'activités pouvant avoir recours à ce nouvel outil. Nous pourrions classer *America's Army* dans les jeux sérieux de type **militaire** et **marketing** en raison de sa volonté de promouvoir l'armée des Etats-Unis.

*Tactical Language & Culture* [13] est un jeu initié en 2003 comme un projet de recherche au laboratoire *Southern California's Information Sciences Institute* sous le financement de DARPA (*Defence Advanced Research Projects Agency*). Son but est d'enseigner les langues et cultures étrangères. Actuellement, ce jeu propose des enseignements pour l'arabe d'Irak (Tactical Iraqi), le pashtou d'Afghanistan (Tactical Pashto), et le français du Sahel Africain (Tactical French) en vue de préparer les militaires aux opérations dans ces régions. *Tactical Language & Culture* est un jeu basé sur un jeu de rôle. Pour permettre une communication immersive, le joueur interagit avec un tuteur virtuel qui évalue l'apprenant et lui fournit des retours sur ces erreurs. Nous pourrions classer *Tactical Language & Culture* dans les jeux sérieux de type **militaire** et **éducatif/formatif**.

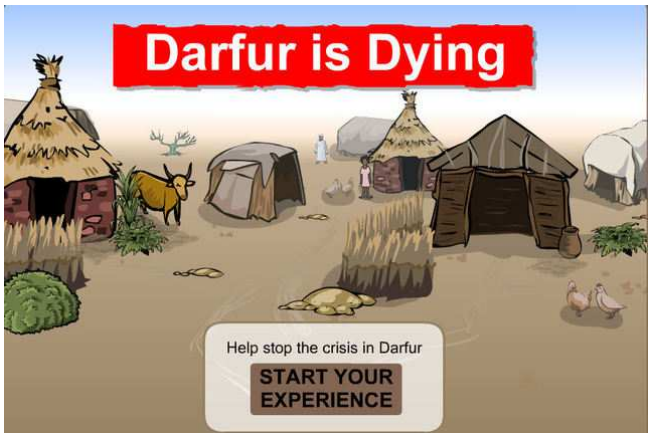



FIGURE 1.5 – *Darfur is dying* : un jeu sérieux militant.

*Darfur is Dying* [14] (voir figure 1.5) est un jeu développé avec la fondation *Reebok Human Rights* et le groupe *International Crisis*. Le but de ce jeu est de faire prendre conscience au grand public des conséquences de la crise du Darfour sur la population. Le joueur contrôle l'un de ces habitants et doit maintenir en fonctionnement un camp de réfugiés face aux possibles attaques

13. `http://www.tacticallanguage.com/` accédé le 25 Mars 2010.
14. `http://www.darfurisdying.com/` accédé le 25 Mars 2010.

des miliciens. L'objectif de ce jeu est de toucher un maximum de personnes, par conséquent, c'est un mini jeu gratuit et accessible en ligne depuis le mois d'Avril 2006. Nous pourrions classer *Darfur is Dying* dans les jeux sérieux de type **militant** et **informatif**.

*Moonshield* [15] (publié en Octobre 2008) est un jeu de gestion/stratégie créé par la société KTM Advance. Il propose au joueur, dans un contexte d'anticipation proche, de mettre en œuvre les technologies de la société Thales pour préserver la terre d'une pluie d'astéroïdes qui pourrait détruire toute civilisation. C'est un jeu gratuit et accessible en ligne. Nous pourrions classer *Moonshield* dans les jeux sérieux de type **marketing** et **informatif**.

**Quelques résultats d'utilisation de jeux sérieux en contexte réel :** Rick Blunt [Blu06] a analysé trois jeux (*Industry Giant II*, *Zapitalism* et *Virtual U*) et leurs impacts sur les résultats d'étudiants suivant des formations en économie. Il a étudié cette influence en fonction du sexe, de l'origine ethnique et de l'âge des participants. Il en conclut que dans tous les cas les jeux sérieux apportent un gain significatif à la formation (voir figure 1.6).

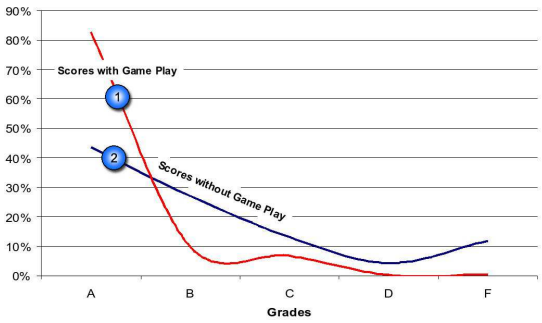



FIGURE 1.6 – Influence du jeu *Zapitalism* sur les performances des étudiants.
*Avec le jeu (courbe 1), 83% des personnes obtiennent la note maximale A. Sans l'utilisation du jeu (courbe 2), il y a davantage de mauvais résultats (D et F).*

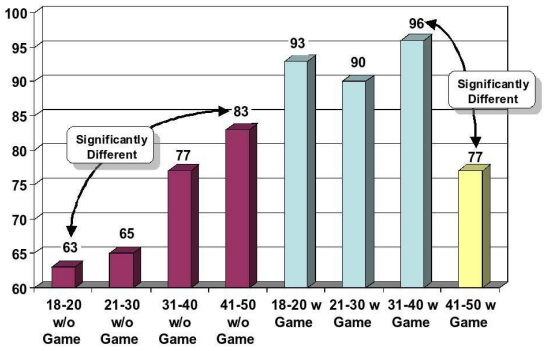



FIGURE 1.7 – Utilité du jeu *Virtual U* en fonction de l'âge des participants.
*Les scores sont moins bon sans utiliser le jeu (les quatre premiers bâtons) qu'avec le jeu (les quatre bâtons suivants) à l'exception de la tranche d'âge 41-50 ans (le dernier bâton).*

On notera une exception intéressante pour les personnes de 41 à 50 ans (voir figure 1.7) qui obtiennent de moins bons résultats. Blunt note à ce propos que ces données renforcent la perception selon laquelle les personnes de cette génération apprennent mieux avec des méthodes traditionnelles avec lesquelles elles se sont construites. Dans leur cas, l'utilisation d'un jeu

15. `http://www.moonshield.com/` accédé le 25 Mars 2010.

vidéo représente une difficulté supplémentaire aux obstacles épistémologiques inhérents à la discipline économique.

### 1.2.2 Composantes d'un jeu sérieux

Les quelques jeux sérieux présentés dans la section 1.2.1 utilisent le divertissement pour atteindre différents objectifs : *America's Army* est un **outil de recrutement** ; *Tactical Language & Culture* a pour but d'**enseigner** ; *Darfur is dying* tente de **sensibiliser** ; *Moonshield* est utilisé à des fins de **communication**. Dans tous les cas, ces jeux ont su habilement équilibrer les deux composantes : « jeu » et « sérieux ». En effet, un jeu sérieux n'est pas seulement un logiciel, un scénario et une interface graphique. Il implique une pédagogie pour atteindre son objectif. Cet ajout, sans subordonner l'histoire, rend le jeu sérieux. La dimension pédagogique est donc un point important qui doit être intégrée dès les premières phases de conception du jeu sérieux.

Dans un contexte de jeu sérieux éducatif, Johnson *et al.* [JVM05] détaillent l'ensemble des composantes propres aux jeux vidéo et précieuses pour maximiser l'apprentissage :

- le *gameplay* est l'une des principales caractéristiques d'un jeu réussi. Johnson *et al.* définissent le *gameplay* comme toutes les activités et stratégies employées par les concepteurs de jeux pour obtenir et garder le joueur engagé et motivé durant tout le jeu. Le *gameplay* ne résulte pas que du graphisme. Deux aspects du *gameplay* sont importants : engager le joueur à chaque instant et relier chaque action aux objectifs futurs ;
- un *feedback* (retour d'information) doit être généré par le jeu suite aux actions du joueur pour lui permettre de chercher à améliorer ses performances. Ces retours sont très importants pour les jeux sérieux, car ils indiquent au joueur s'il réussit ou non ;
- une interface simple, bien définie, qui supporte les interactions entre le joueur et le jeu (l'affordance) est gage de qualité. Par exemple, elle peut suggérer ou guider les actions de l'utilisateur. Ces ajouts d'informations ne correspondent pas à une scène réelle, mais sont nécessaires pour maintenir une interaction fluide entre le joueur et le jeu (voir figure 1.8) ;
- les difficultés à surmonter doivent être adaptées à l'expérience du joueur. S'il y a un trop grand décalage entre les capacités du joueur et la difficulté demandée, le jeu perdra de son intérêt. Il est donc souhaitable de proposer une version allégée du jeu réel où la complexité du *gameplay* est limitée. Ceci permet au joueur de développer ses compétences avant de rencontrer les défis du jeu complet ;

– dans les jeux sérieux modernes, l'utilisation du scénario est fondamentale pour maintenir l'intérêt du joueur et l'encourager à s'identifier au personnage ;
– enfin, un bon jeu doit être ludique. Cet aspect permet de maintenir l'intérêt et une attitude positive du joueur.

FIGURE 1.8 – Exemple d'affordance dans un jeu vidéo.
*La flèche au dessus du personnage aide le joueur dans la compréhension de la scène.*

En vue de comprendre pourquoi un joueur est motivé par l'environnement de jeu, Siang et Rao [SR03] ont adapté la pyramide des besoins de Maslow. Cette hiérarchie est divisée en sept niveaux où les premiers servent de base aux niveaux supérieurs. Les sept niveaux par ordre de priorité sont les suivants : (1) Le **besoin de règles**, les joueurs recherchent des informations pour comprendre les règles de base structurant le jeu ; (2) Le **besoin de sûreté**, les joueurs ont besoin de trouver de l'aide sur le fonctionnement du jeu ; (3) Le **besoin d'appartenance**, les joueurs ont besoin de s'approprier le jeu pour se sentir capable d'atteindre les objectifs ; (4) Le **besoin d'estime**, les joueurs ont besoin d'être valorisés par le jeu (*feedback*, progression, score, compétition, etc.) ; (5) Le **besoin de connaître et de comprendre**, les joueurs ont besoin de découvrir les informations/bonus cachés et de les mettre en relation en vue de les réinvestir en situation de jeu ; (6) Le **besoin d'esthétique**, les joueurs ont besoin de beaux graphismes, d'effets visuels, d'une musique appropriée, d'effets sonores, etc. ; (7) Le **besoin d'auto accomplissement**, les joueurs veulent être capables de projeter leur créativité et imagination dans le jeu sous contrainte du respect des règles.

Cette hiérarchie des besoins permet de garder à l'esprit quelques principes lors de la conception d'un jeu sérieux. Par exemple, des graphismes saisissants ne sauveront pas à eux seuls un

mauvais *gameplay*. Siang et Rao précisent que si un joueur ne comprend pas les règles du jeu dans les premières minutes, il risque simplement de s'en désintéresser. Mais lorsque le jeu sérieux est correctement équilibré, des résultats intéressants peuvent être obtenus.

Outre toutes ces caractéristiques, Wolf et Alvarez (voir section 1.1.2) soulignent l'importance de fixer les critères de classement selon l'interactivité. Dans ce sens le concept de « jeu sérieux » ne caractérise pas un genre de jeu particulier mais est une composante des genres de jeux existants. En reprenant la classification de *GameSpot*, il est alors possible d'imaginer des jeux sérieux d'aventure, de plates-formes, de stratégie temps réel, etc. La combinaison du concept de « jeu sérieux » avec le cas des jeux de simulation pose alors le problème de sa différence avec les simulateurs.

## 1.3 Généralités sur les simulateurs

Le dictionnaire TLFi définit un simulateur comme un « *dispositif expérimental ou programme d'ordinateur qui permet de reproduire artificiellement le fonctionnement réel d'un appareil, d'une machine, d'un système, d'un phénomène, à des fins d'étude, de démonstration ou d'explication* ». Le point commun de tout simulateur est donc de **reproduire artificiellement un fonctionnement réel**. La forme et la finalité du simulateur dépendent de l'objectif à atteindre. Par exemple, les simulateurs de vol sont composés d'un poste de pilotage réel pour préparer les pilotes dans des conditions réalistes alors que les logiciels de simulation météorologique sont purement logiciels et sont utilisés à des fins d'études et de prévision. Dans ces deux exemples, la reproduction de la réalité est la composante première que doivent respecter les simulateurs.

L'étude de la simulation et de la modélisation se rapporte à la représentation du temps et à l'état de la simulation dans un modèle. Sur cette base, les simulateurs sont traditionnellement classés selon le modèle qu'ils adoptent [NWFR06]. Un simulateur basé sur le modèle de *Monte Carlo* présente les différents états de la simulation sans représentation explicite du temps. Un simulateur basé sur un modèle *d'événements discrets* indique les changements d'état à des instants précis de la simulation. Les simulateurs basés sur une *simulation continue* décrivent les changements d'état en continu à travers le temps. Les modèles combinant *événements discrets et simulation continue* permettent d'appliquer les deux techniques dans un même simulateur. Les simulateurs basés sur un modèle de *simulation hybride* contiennent généralement un sous-modèle analytique dans un modèle d'événements discrets.

### 1.3.1 Domaines d'application

Les simulateurs et les technologies de simulation sont utilisés dans de nombreux domaines scientifiques et industriels. Narayanasamy *et al.* [NWFR06] dressent un éventail des domaines d'applications traités par les simulateurs :

- **l'analyse de systèmes** en vue de comprendre leur fonctionnement et leurs concepts ;
- **l'éducation** pour offrir un environnement qui promeut le développement de modèles mentaux chez l'apprenant ;
- **l'apprentissage et l'acceptation** pour répondre aux questions relatives aux systèmes qui attendent un certain nombre de prérequis et aux sous-systèmes qui contribuent de manière significative à l'amélioration des performances de systèmes plus larges ;
- **la recherche** pour recréer des environnements artificiels afin de tester les composantes du système ou tester le comportement d'un ou plusieurs individus dans cet environnement à des fins de comparaisons ou de catégorisation ;
- **l'armée** pour supporter plusieurs objectifs comme l'entraînement, l'analyse, l'apprentissage, la répétition de missions et l'évaluation.

### 1.3.2 Historique

Cet historique retrace l'évolution des simulateurs interactifs et plus particulièrement des simulateurs d'entraînement basés sur une simulation continue car ce sont les plus proches des jeux vidéo. Ces simulateurs sont dits « interactifs » car ils intègrent l'homme dans la boucle de simulation. Les systèmes de réalité virtuelle sont couramment utilisés dans les simulateurs pour positionner l'utilisateur en situation d'interaction avec le monde virtuel. Un dialogue doit donc s'établir entre l'homme et la machine, c'est la boucle « perception, cognition, action » (voir figure 1.9) définie dans le « Traité de la réalité virtuelle » [FMT06, p. 9]. L'utilisateur perçoit le monde virtuel au travers d'interfaces sensorielles. Les périphériques permettent à l'utilisateur de réaliser des activités qui sont transmises au calculateur. Celui-ci les interprète et modifie l'environnement si besoin, puis restitue les informations sensorielles à l'utilisateur.

- **1910** : Léon Levavasseur conçoit un des premiers simulateur de vol, le « tonneau Antoinette ». Le poste de pilotage, monté sur rotule, est actionné manuellement en lacet, roulis et tangage (voir Figure 1.10) ;
- **1915** : témoignage photographique d'un simulateur équestre mécanique en bois (voir figure 1.10) ;

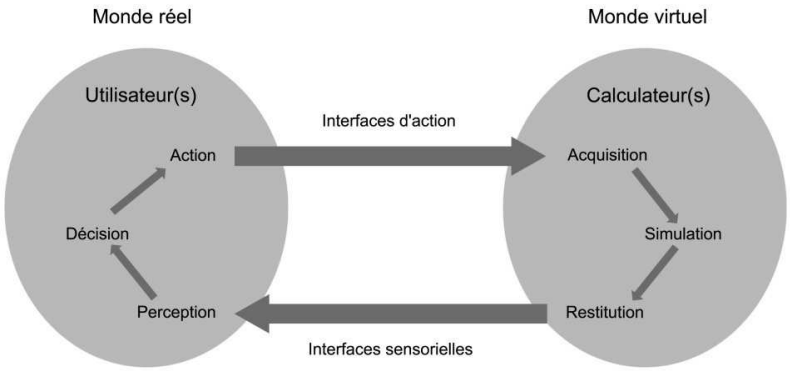



FIGURE 1.9 – Boucle « perception, cognition, action » en communication avec le monde virtuel.

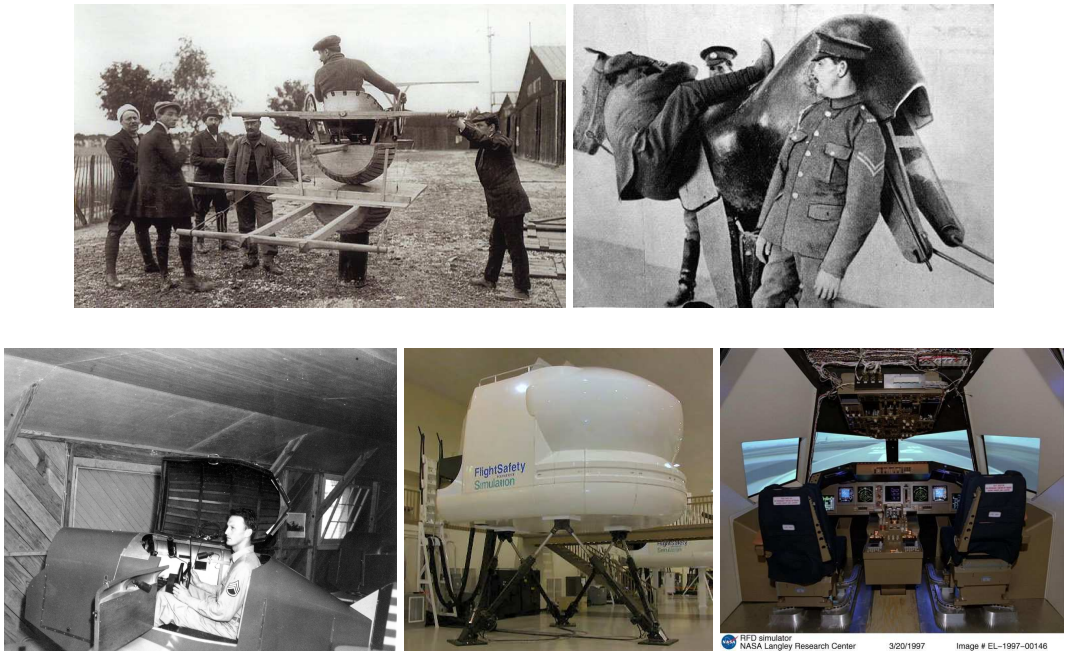



FIGURE 1.10 – Simulateurs.
*De gauche à droite et de haut en bas : le « tonneau Antoinette »de Léon Levavasseur (1910), simulateur équestre mécanique en bois (1915), le « Link Trainer » d'Edwin Link (1929), un simulateur de vol civil à mouvement « six axes », poste de pilotage expérimental d'un simulateur NASA.*

– **1929** : Edwin Link conçoit le « *Link Trainer* » (voir Figure 1.10). Ce simulateur utilisé lors de la seconde guerre mondiale marque le début de l'utilisation de simulateurs dans un cadre de formation. Dès 1940, les ordinateurs seront intégrés aux simulateurs pour résoudre les équations de vol. Aujourd'hui, l'utilisation de simulateurs pour la formation

des pilotes d'avion (civils ou militaires) s'est démocratisée. Les simulateurs les plus récents tendent vers un réalisme de plus en plus poussé avec l'amélioration des systèmes de mouvements (six degrés de liberté) et des systèmes de visualisation (images hautes définitions, miroir courbé) ;

– **1983** : SIMNET (*SIMulator NETworking*) a pour objectif de fournir un monde virtuel permettant de simuler un champ de bataille où cent mille entités (simulateurs différents et délocalisés) peuvent évoluer. L'application doit pouvoir fonctionner en temps réel, en répartition totale et être peu coûteuse par rapport au coût d'une simulation grandeur nature. Le problème clé du projet est de relier, à travers un réseau, les différentes entités afin de créer un champ de bataille virtuel cohérent. Il sera opérationnel en 1990. Trois ans plus tard, DIS (*Distributed Interactive Simulation*) est le successeur de SIMNET. Avec ce système, un simple ordinateur pourvu d'une carte réseau et pouvant gérer le protocole de communication de DIS peut prendre part à la simulation ;
– **1984** : Inspiré par les simulateurs de vol, le centre de recherche suédois VTI (*Väg och transportforskningsinstitutet*) à Linköping conçoit l'un des premiers simulateurs de conduite automobile. De nombreux constructeurs automobiles ont développé, depuis, leurs propres simulateurs en vue d'étudier le comportement des automobilistes au volant de leurs voitures. Le plus grand, actuellement en fonctionnement, a été conçu par Toyota. Il est installé au centre technique de Higashifuji (2007). Ce simulateur de 4,5 mètres de haut et 7,1 mètres de diamètre peut se déplacer sur 700 $m^2$ et s'incliner jusqu'à 25 degrés pour recréer des accélérations de 0,5g.

# Conclusion

La croissance de ventes des jeux vidéo (voir figure 1.11) témoigne de la démocratisation de ce type de divertissement. Sue Blackman [Bla05] fait une synthèse de ce domaine d'activités. Les investissements conférés à cette industrie, pour répondre à la demande du marché, permettent le développement d'outils et de bibliothèques facilitant et améliorant la conception des jeux vidéo. Les moteurs graphiques des jeux vidéo, de plus en plus perfectionnés, peuvent même être utilisés pour des applications autres que le jeu, car ils proposent des rendus temps réels et des moteurs physiques réalistes. Des applications d'entraînement, de visualisation interactive et de simulation de situation utilisent largement les technologies des jeux vidéo.

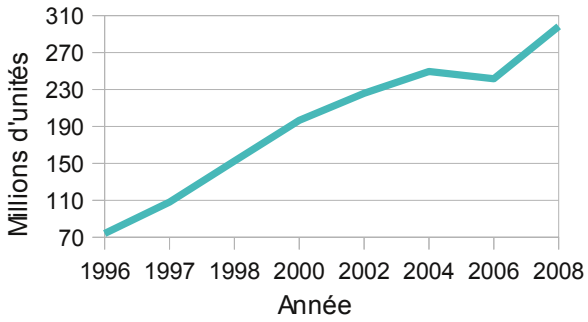



FIGURE 1.11 – Croissance de ventes des jeux vidéo de 1996 à 2008.
*Données extraites de l'enquête de l'ESA [Ass09].*

*America's Army* est un bon exemple de logiciel considéré comme un jeu vidéo quand il est téléchargé et joué pour sa composante ludique, comme un jeu sérieux quand il est utilisé comme outil de recrutement et comme un simulateur quand il est intégré dans les stages d'entraînement militaire. Il est alors possible de considérer les jeux sérieux comme des intermédiaires entre les jeux vidéo et les simulateurs. Ils possèdent la composante ludique des jeux vidéo tout en proposant, parfois, un objectif de formation ou d'apprentissage propre aux simulateurs. Toutefois, dans certains cas les frontières entre ces trois types d'applications restent floues.

La recherche conduite dans ce mémoire se concentre sur la conception, la réalisation et l'analyse d'un jeu sérieux de stratégie temps réel pour l'apprentissage de la programmation. Dans ce cadre, il convient de présenter en détail ce type de jeu.

# 2

# Jeux de stratégie temps réel (STR)

Parmi les classifications de jeux vidéo présentées dans la section 1.1.2, celle de *GameSpot* définit trois types de jeux de stratégie : les jeux de stratégie temps réel (STR), les jeux de stratégie au tour par tour et les autres jeux de stratégie [16]. Le choix de l'utilisation d'un STR comme base au jeu sérieux est motivé dans un premier temps par sa complexité qui en fait un environnement intéressant pour la mise en pratique d'exercices de programmation. D'autre part, les jeux de stratégie représentent un genre de jeu populaire sur ordinateur comme le confirme les chiffres de l'ESA [Ass09]. Celle-ci positionne les jeux de stratégie comme le type de jeu le plus vendu sur ordinateur aux États-Unis avec 34,6% des ventes sur l'année 2009. Enfin, l'existence de moteur de jeu de STR à code source ouvert représente une base utile à la réalisation d'un jeu sérieux. Ce chapitre définit donc le jeu de STR à travers l'analyse de *Dune 2* [17] (un jeu ayant fortement concouru à l'émergence de ce genre), présente ensuite à travers un court historique les innovations importantes apportées par de nombreux nouveaux titres et décrit enfin quelques projets à code source ouvert témoins de l'effervescence induite par les jeux de STR.

16. Le type « autres jeux de stratégie » regroupe l'ensemble des jeux de stratégie qui n'entrent pas directement dans l'un des deux autres types (STR ou tour par tour). À titre d'exemple, la série des *Caesar* (*Sierra Entertainment*) et des *Settlers* (*Blue Byte Software*) sont classés par *GameSpot* dans cette catégorie.

17. Westwood Studios, 1992

## 2.1 Définition générale des jeux de STR fondée sur le jeu *Dune 2*

*Dune 2* est une adaptation en jeu vidéo du roman de Frank Herbert, Dune. Dans ce jeu, trois Maisons (les Atréides, les Harkonnens et les Ordos) s'affrontent pour le contrôle de la planète Arrakis. Le joueur incarne le rôle d'un commandant appartenant à l'une des trois Maisons. Il devra livrer bataille contre les autres Maisons à travers plusieurs missions. Son objectif consiste à développer une base militaire en vue de lever une armée pour éliminer les troupes et bases ennemies.

Pour expliciter le fonctionnement de *Dune 2*, une analogie avec le jeu d'échec s'avère pertinente. En effet, les échecs font partie des jeux de stratégie, il est donc possible d'effectuer un certain nombre de comparaisons entre les échecs et les jeux de STR. Le jeu d'échec est composé d'un plateau et d'un ensemble de pièces. Le plateau est de topographie plane composé de 64 cases où chacune d'elle peut être repérée par un chiffre compris entre 1 et 8 (lignes) et une lettre comprise entre « a » et « h » (colonnes). Chaque pièce possède un certain nombre d'attributs tels que son apparence, sa position et son mode de déplacement.

De même, *Dune 2* propose au joueur de réaliser les parties sur un certain nombre de cartes géographiques (équivalentes au plateau du jeu d'échec) avec un ensemble d'unités de départ (équivalentes aux pièces du jeu d'échec). Au même titre que les pièces du jeu d'échec se déplacent sur le plateau de jeu, les unités de *Dune 2* évoluent sur une carte où les positions des unités sont repérées par des coordonnées x et y. La carte peut contenir du relief, différents types de sols ainsi que des éléments de décors tels des impacts d'obus, des bâtiments, des ressources, etc. Généralement, les ressources jouent un rôle important dans les jeux de STR, car, présentes en quantités limitées (l'épice pour *Dune 2*), elles sont nécessaires et convoitées par tous les joueurs. Les unités sont dotées également d'un certain nombre d'attributs tels qu'une apparence, une position, un mode de déplacement, mais aussi un capital santé, un champ de vision, un coût, une force, etc. L'ensemble de ces attributs caractérise le type d'une unité.

Chaque nouveau jeu de STR tente d'introduire des innovations afin de surprendre les joueurs. Ces innovations peuvent porter sur la qualité des graphismes, l'univers du jeu (médiéval, futuriste, etc.), le scénario, les caractéristiques des unités, le nombre d'unités, etc.

À partir de la définition du jeu vidéo de Mickael Zyda présentée dans la section 1.1 nous avons identifié deux points clé pour caractériser un jeu vidéo à savoir définir **les règles** et **les buts** du jeu.

### 2.1.1 Règles

Dans *Dune 2* quatre règles fondamentales ont été identifiées : le « temps réel », le « brouillard de guerre » le « contrôle indirect » et « l'arbre des technologies ». Le « temps réel » est opposé au « tour par tour » utilisé aux échecs par exemple où, chaque joueur, l'un après l'autre, modifie l'état du jeu à sa convenance. Dans un jeu en « temps réel », chacun est libre de réaliser une action à n'importe quel instant. Par conséquent le joueur doit être capable d'adapter rapidement sa stratégie à un environnement hautement dynamique.

Le « brouillard de guerre » cache les parties de l'environnement non encore explorées par le joueur. Les zones visibles dépendent en effet des positions de ses unités sur la carte. En conséquence, les unités de l'adversaire sont cachées tant qu'elles n'entrent pas dans une zone déjà découverte par le joueur. La figure 2.1 illustre le « brouillard de guerre » tel qu'il était présent dans *Dune 2*.

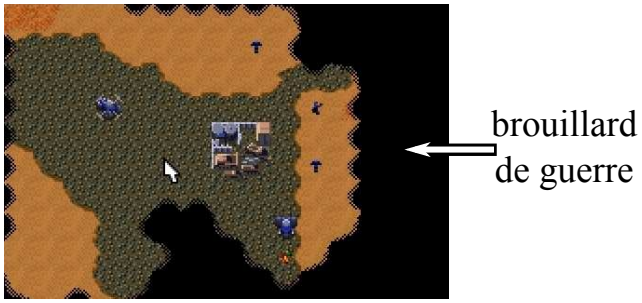
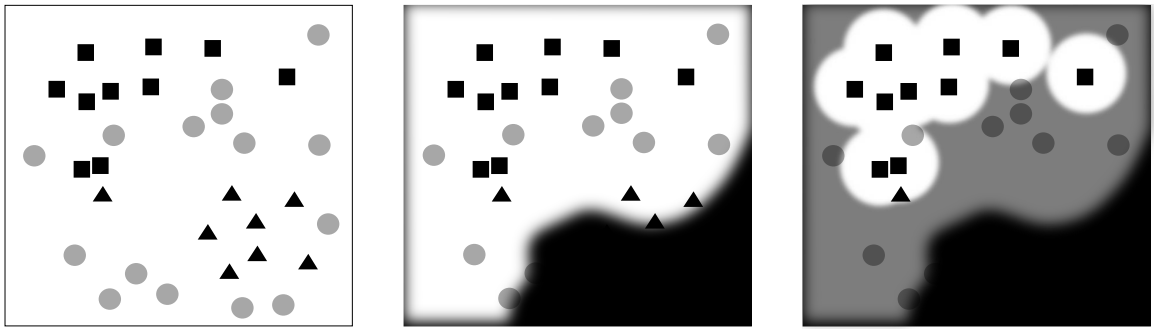



FIGURE 2.1 – Représentation du « brouillard de guerre » dans *Dune 2*.

Le principe de « brouillard de guerre » a été amélioré par la suite en prenant en compte le « champ de vision » de chaque unité. Dans ce contexte, pour être visibles, les unités de l'adversaire doivent pénétrer le champ de vision d'une unité contrôlée par le joueur et ne pas simplement entrer dans une zone déjà explorée et non surveillée. La figure 2.2 montre la différence de perception d'une situation de jeu en fonction de la présence du « brouillard de guerre » et de la prise en compte du « champ de vision ». La figure 2.2a présente la scène de jeu. Elle contient les unités du joueur et de son adversaire ainsi qu'un certain nombre d'éléments de

décor immobiles. La figure 2.2b illustre l'influence du brouillard de guerre. Il représente les zones non explorées par le joueur et cache les unités adverses et les éléments de décor situés en dessous. La figure 2.2c ajoute les champs de vision des unités du joueur. Dorénavant, seules les unités adverses sous surveillance sont visibles. Les éléments de décor immobiles précédemment découverts sont toujours connus et affichés.



(a) Scène de jeu.

(b) Scène de jeu avec brouillard de guerre.

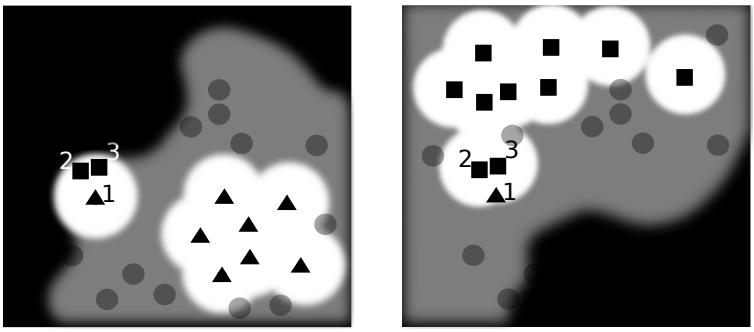
(c) Scène de jeu avec brouillard de guerre et champs de vision.

■ : Entité contrôlée par le joueur.
▲ : Entité contrôlée par son adversaire.
● : Élément de décor immobile.

FIGURE 2.2 – Perception d'une situation de jeu en fonction de la présence du « brouillard de guerre » et de la prise en compte du « champ de vision ».

Dans un contexte multijoueur, chaque joueur évolue dans un environnement mal connu qu'il doit explorer. La Figure 2.3 illustre l'influence du « brouillard de guerre » sur la perception d'une même partie. Dans cette situation de jeu, le joueur A (voir figure 2.3a) contrôle les unités triangulaires et a ordonné à l'une d'entre elles (numérotée 1) de s'éloigner vers la gauche. En se déplaçant, elle détecte deux unités du joueur B (numérotées 2 et 3). Inversement le joueur B (voir figure 2.3b) qui contrôle les unités rectangulaires (dont celles numérotées 2 et 3) voit apparaître l'unité du joueur A (numérotée 1) passant à proximité de ses unités.

Le « contrôle indirect » permet au joueur de définir un ordre pour une ou plusieurs de ses unités. Celles-ci vont alors tenter de l'exécuter sans que le joueur ait besoin de les contrôler directement. Celui-ci peut alors commander d'autres unités en parallèle. Ce mode de contrôle permet également de gérer les conflits inhérents au « temps réel ». En effet, si deux unités ont reçu l'ordre d'atteindre une même position, le jeu résoudra le conflit et déterminera automatiquement celle qui atteindra la position finale et celle qui abandonnera son action.




(a) Écran de jeu du joueur A. (b) Écran de jeu du joueur B.

▲ : Entité contrôlée par le joueur A.
■ : Entité contrôlée par le joueur B
● : Élément de décor immobile.

▭ : Zone explorée sous surveillance (champ de vision).
▬ : Zone explorée sans surveillance.
▬ : Zone non explorée (brouillard de guerre).

FIGURE 2.3 – Influence du « brouillard de guerre » sur la perception de deux joueurs prenant part à une même partie.

Enfin, « l'arbre des technologies » représente une architecture abstraite des différents éléments du jeu. En début de partie, le joueur possède un ensemble restreint d'unités et de structures qui vont lui permettre d'en construire de nouvelles et de débloquer ainsi de nouveaux éléments. La maîtrise de « l'arbre des technologies » est une part non négligeable des jeux de STR car elle permet à un joueur expérimenté de rapidement accéder à une ou plusieurs technologies nécessaires à la mise en œuvre de sa stratégie.

### 2.1.2 Buts

Traditionnellement les jeux de STR proposent deux modes de jeu : la **campagne** et l'**escarmouche**. Le mode campagne a pour objectif de séduire le joueur en vue de lui apprendre à jouer. Une campagne est divisée en missions qui introduisent progressivement le contenu et la complexité du jeu. Dans ce mode de jeu, le joueur devra atteindre des objectifs en lien avec le déroulement des évènements, comme atteindre une position, résister aux attaques de l'adversaire ou construire une unité particulière.

Le mode escarmouche (non présent dans *Dune 2*) a pour objectif d'étendre la vie du jeu en permettant au joueur de remporter des défis contre d'autres joueurs ou l'ordinateur. Dans ce mode de jeu, le but consiste, en général, à éliminer toutes les unités de l'adversaire. Eliminer une unité revient à amener son capital santé à une valeur nulle. Pour ce faire, le joueur doit ordonner à l'une de ses unités d'attaquer l'unité cible. Ces deux unités vont alors engager un combat et produire plusieurs assauts jusqu'à ce que l'une d'entre elles soit éliminée. Chaque assaut réduit l'énergie de l'adversaire du nombre de points de dégâts que peut infliger l'attaquant.

En raison du « brouillard de guerre », le joueur doit dans un premier temps ordonner à ses unités mobiles d'aller explorer la carte en indiquant pour chacune d'elles une position à atteindre. Lorsque l'adversaire est découvert (*i.e.* lorsque l'une des unités du joueur est suffisamment proche d'une unité de l'adversaire pour la découvrir) le joueur peut alors ordonner à ses unités d'engager le combat.

Ces quelques règles et buts posés par *Dune 2* ont été améliorés par la suite avec les nouvelles générations de jeux de STR.

## 2.2 Historique

De nombreux jeux de stratégie en temps réel ont existé avant *Dune 2* à l'image de *Stonkers* [18] (1983), *The Ancient Art of War* [19] (1984) ou encore *Populous* [20] (1989). L'ensemble de ces jeux peuvent être considérés comme les précurseurs des jeux de STR actuels et ont certainement inspiré en leur temps les concepteurs de *Dune 2*. Ce dernier est néanmoins considéré comme le « père » des jeux de STR car les règles et buts issus de ce jeu semblent être, encore aujourd'hui, représentatifs de la majorité des jeux de STR. Toutefois, certains ne reprennent pas en totalité les principes posés par *Dune 2*, ce qui n'affecte en rien leur appartenance à cette famille de jeu. Bien au contraire, cette diversité contribue à renouveler et à faire évoluer les jeux de STR. Pour illustrer ces évolutions, l'historique suivant présente, par ordre chronologique, quelques jeux vidéo ayant participés à l'émancipation des jeux de STR depuis la sortie de *Dune 2* en 1992.

– **1994** : *Warcraft* développé par *Blizzard Entertainment* est le premier titre d'une série ayant fortement participé à démocratiser les jeux de STR. Il a initié l'aspect multijoueur

18. Imagine Software, 1983
19. Evryware, 1984
20. Bullfrog, 1989

à travers le mode escarmouche et a introduit la fonctionnalité de pouvoir sélectionner plusieurs unités et de les commander simultanément.

- **1995** : *Command & Conquer* développé par *Westwood Studio* est également le premier jeu d'une longue série. Il a notamment développé le principe de gestion de groupes d'unités.
- **1996** : *Warcraft 2* améliore le « brouillard de guerre » en exploitant le « champ de vision » de chaque unité.
- **1997** : *Total Annihilation* développé par *Cavedog Entertainment* est le premier STR à représenter les unités et les cartes de jeu par des maillages 3D. Il introduit également un moteur physique réaliste dans un jeu de STR. Dans cette même année, *Age of Empires* développé par *Ensemble Studios* met l'accent sur « l'arbre des technologies » et propose une architecture plus complexe inspirée des jeux de stratégie en « tour par tour » tels que *Civilization* [21].
- **1998** : *Starcraft* développé par *Blizzard Entertainment* est le jeu de STR à avoir remporté le plus grand succès. Très peu de jeux vidéo ont eu une durée de vie supérieure à 10 ans. Un circuit de compétitions internationales (incluant des championnats professionnels) s'est organisé autour de ce jeu. Dans cette même année, le jeu *Battlezone* développé par *Activision* tente un mélange des jeux de STR et des jeux de tir subjectif.
- **1999** : *Homeworld* développé par *Relic Entertainment* introduit un environnement tridimensionnel et autorise une liberté de mouvement en tout point de l'espace.
- **2001** : *Cossacks* développé par *GSC Game World* porte le nombre d'unités contrôlables à plus de 8000.
- **2002** : *Warcraft 3* développé par *Blizzard Entertainment* intègre certains aspects du jeu de rôle dans les jeux de STR.

Les années 90 ont donc vu la naissance et les évolutions majeures des jeux de STR actuels. Historiquement, ce type de jeu est principalement distribué sur ordinateur. La première raison trouve son origine dans l'interaction. Ce type de jeu nécessite un dispositif de pointage rapide et précis. Contrairement au périphérique d'entrée historique des consoles de jeu (la manette), la souris de l'ordinateur se prête bien mieux à ces spécifications. La deuxième raison vient de la forte composante « multijoueur » non disponible sur console avant la sixième génération (1998). Cet aspect « multijoueur » est donc une composante importante des jeux vidéo et a fortement contribué au succès de certains jeux de STR comme *Starcraft*. La conception et la mise au point

21. MicroProse, 1991

de ce mode de jeu implique des choix importants dans le processus de réalisation. A ce titre, il convient de présenter les différents types d'architectures réparties.

## 2.3 Architectures réparties

L'architecture répartie détermine l'organisation physique des différentes machines ainsi que leur rôle. L'emplacement des données et leur gestion dépendent également de cette architecture. Les deux grands principes s'articulent autour de la présence ou non d'un serveur, il sera alors question de systèmes centralisés et de systèmes sans serveur.

### 2.3.1 Systèmes centralisés

Ils s'organisent autour de deux entités : le serveur (une machine généralement très puissante en termes de capacités d'entrée-sortie qui fournit des services) et le client (une machine où s'exécute un programme qui communique avec le serveur pour récupérer des informations). Ces systèmes peuvent être découpés en trois sous groupes : l'architecture client-serveur centralisée, l'architecture client-serveur distribuée et l'architecture client-serveur distribuée avec plusieurs serveurs :

- architecture client-serveur centralisée (voir figure 2.4) : la base de données représentant le monde virtuel est entièrement gérée par un seul serveur. Lorsqu'un client se connecte à ce serveur, il exige des ressources (mémoire, CPU (*Central Processing Unit*), bande passante) mais plus le nombre de clients augmente plus cette exigence sera difficile à satisfaire. Ce modèle d'architecture est évidemment peu extensible et permet de gérer au mieux une cinquantaine d'utilisateurs en même temps dans le cas de communications purement textuelles (MUD par exemple) ;
- architecture client-serveur distribuée : la base de données représentant le monde virtuel est répartie sur les différents clients. Seules les communications entre les clients sont gérées par le serveur. Cette amélioration réduit l'utilisation de la bande passante, mais le serveur reste un goulot d'étranglement ;
- architecture client-serveur distribuée avec plusieurs serveurs : Pour gérer ce problème plusieurs sont mis en place et communiquent. Ces serveurs sont organisés et peuvent gérer les clients de diverses façons : partitionnement basé sur la localisation physique ou virtuelle des clients. Concernant la localisation physique, chaque serveur doit avoir

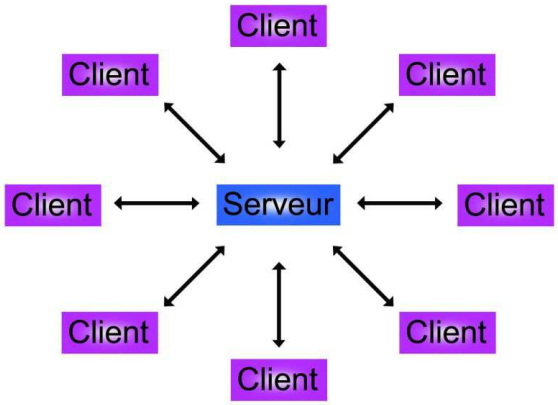



FIGURE 2.4 – Architecture client-serveur centralisée.

une image du monde virtuel dans son ensemble, ce qui est donc difficilement extensible. La localisation virtuelle est plus facilement extensible car chaque serveur ne gère qu'une partie du monde virtuel. Cependant, comme les clients sont inégalement répartis dans le monde virtuel, certains serveurs peuvent être rapidement surchargés. Il devient donc nécessaire de mettre en place une gestion des clients en vue d'équilibrer la charge des différents serveurs. Le problème de distribution de charges est résolu par des algorithmes statiques, dynamiques et adaptatifs où les algorithmes adaptatifs sont considérés comme une classe spéciale des algorithmes dynamiques. Les algorithmes de distribution de charges dynamiques sont encore classés en algorithmes de partage de charges et en algorithmes de mise en correspondance de charges. L'équilibrage des charges n'est pas la seule difficulté à surmonter dans ce type d'architecture. En effet, l'augmentation du nombre de serveurs rend le maintient d'une base de données cohérente plus difficile. D'autre part, les messages envoyés par les clients peuvent traverser plusieurs serveurs avant d'aboutir au destinataire ce qui a pour conséquence d'augmenter la latence [22] et donc de réduire le niveau d'interactivité. Cependant, cette architecture a pour avantage de diminuer l'impact de l'ajout de nouveaux clients. L'architecture client-serveur distribuée avec plusieurs serveurs est couramment utilisée pour supporter les MMOG (Duong et al. [DZ03] en présentent un exemple). Steinkuehler [Ste04] qualifie de MMOG tout jeu

22. latence : temps pris par un message pour passer d'un hôte à un autre en lui ajoutant le temps du traitement de l'information émise et reçue

qui satisfait une accessibilité en ligne à un très grand nombre de joueurs et qui assure une persistance du jeu qu'il y ait des joueurs connectés ou pas. En raison de leur grande interactivité, la migration des clients est une tâche très lourde, car des milliers de personnes sont gérées par le jeu.

### 2.3.2 Systèmes sans serveur

Ils sont organisés autour de l'architecture égal à égal (*Peer-to-Peer* ou P2P). Dans cette configuration, chaque ordinateur a le même rôle, il apporte de la mémoire, du temps de calcul et de la bande passante pour gérer les états des données partagées (voir figure 2.5). La base de données représentant le monde virtuel est dupliquée sur les différentes machines qui communiquent les modifications locales à tous les autres ordinateurs.

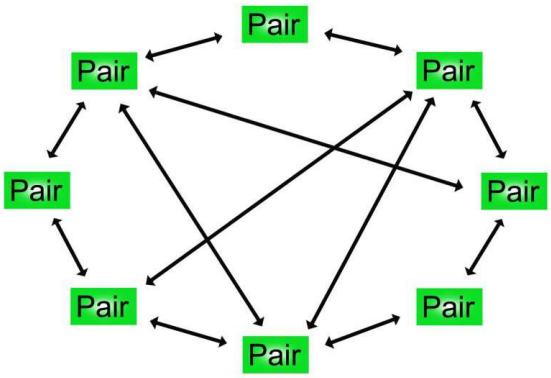



FIGURE 2.5 – Architecture pair à pair distribuée en communication point à point.

Ce type d'architecture présente deux inconvénients majeurs :

– au niveau de l'extensibilité : le nombre de messages de mise à jour augmente proportionnellement avec le nombre de machines. Pour pallier ce type de problème, il est possible de diminuer la fréquence des messages de mise à jour grâce, par exemple, à la technique du *dead-reckoning*. Cette méthode de prédiction consiste à déterminer le comportement d'une entité virtuelle en fonction de ses anciens messages de mise à jour. Dans ce cadre, un nouveau message est envoyer seulement si la prédiction est trop mauvaise par rapport à la situation réelle ;

– au niveau du maintien de la cohérence des copies de la base de données représentant le monde virtuel. De nombreux mécanismes, issus des systèmes d'exploitation distribués, existent pour gérer ce problème, du plus simple : à un instant donné, un seul ordinateur est autorisé à modifier la base de données et ce droit circule de machine en machine (grâce à un anneau à jeton logique par exemple) ; au plus complexe : toutes les machines procèdent à des modifications en précisant leur date d'altération et lorsqu'un conflit se produit la modification la plus ancienne est prise en compte et les autres sont annulées.

Traditionnellement les MMOG sont développés sur une architecture client-serveur, car les serveurs gèrent plus facilement les comptes des joueurs ainsi que les états du jeu. Cependant, cette architecture est moins flexible, plus onéreuse et limite l'extensibilité du nombre de joueurs. Knutsson et al. [KLXH04] proposent une approche basée sur le P2P avec un système grandissant et rétrécissant dynamiquement avec le nombre de joueurs. Trois types de problèmes doivent être résolus pour rendre cette approche pleinement applicable : la performance, la disponibilité (perte d'un nœud et maintient d'un grand nombre de redondance), et la sécurité (protéger les comptes et empêcher la tricherie). Avec cette architecture réseau et pour un même nombre de joueurs par région, le nombre total de joueurs connectés n'a aucune conséquence sur les performances. En revanche, le système est plus sensible aux différences de concentration de joueurs, mais ce problème peut être résolu par le jeu en limitant le nombre de joueurs par région ou en modifiant dynamiquement la taille des régions en fonction de l'affluence des joueurs.

## Synthèse

Chacune de ces deux architectures possède des avantages et des inconvénients, le choix n'est pas évident. Tout dépend de l'application et de ses contraintes en termes d'extensibilité, de fiabilité, de simplicité et d'interactivité.

Le modèle client-serveur est particulièrement recommandé pour des applications nécessitant un grand niveau de fiabilité. Les ressources étant centralisées, il est plus facile de gérer des ressources communes à tous les utilisateurs et d'éviter les problèmes de redondance et de contradiction. La sécurité est accrue, car le nombre de points d'entrée permettant l'accès aux données est moins important. En revanche, le coût est élevé en raison de la mise en place des infrastructures (bâtiments, ordinateurs, climatisations, etc.). De plus, le serveur est le maillon faible de l'architecture client-serveur, étant donné que toute l'application est architecturée autour de lui. La panne d'un serveur peut perturber, voire même bloquer, complètement l'application.

Le P2P, quant à lui, est plus flexible, plus extensible et beaucoup moins onéreux en terme de matériel, mais il est aussi plus difficile de garantir des performances, de la disponibilité et de la sécurité, car à tout moment un nœud responsable de données peut disparaître. Traditionnellement, le mode de jeu multijoueur des STR est implémenté sur une architecture P2P. La publication du code réseau de la série des *Age of Empires* [BT01] en est une illustration.

Concernant les jeux de STR massivement multijoueurs (ou MMORTS - *Massively Multiplayer Online Real-Time Strategy games*), peu d'exemples existent. Il est possible de citer OGame [23] et Saga [24]. Les difficultés liées à cette forme de MMOG portent sur la gestion des joueurs hors ligne et aux problèmes techniques liés à la perception des joueurs de l'environnement. La plupart des MMORTS placent le joueur à la tête d'un empire qu'il doit faire prospérer. Lorsque ce dernier se déconnecte des serveurs, toutes les entités relatives à son empire restent instanciées dans le jeu en raison de la persistance du monde virtuel et se retrouvent sans commandement. Le jeu doit alors intégrer des IA (Intelligences Artificielles) performantes capables de prendre le relais ou instaurer des règles de jeu spécifiques pour protéger les joueurs déconnectés.

Les problèmes techniques liés aux MMORTS viennent du fait que le joueur ne contrôle pas qu'un seul avatar [25] mais des dizaines voire des centaines d'entités simultanément. Chacune d'entre elles perçoit une partie du monde virtuel et augmente par conséquence la quantité d'informations à véhiculer à travers le réseau. Ce phénomène est accentué si le niveau d'interaction du MMORTS tend à se rapprocher de celui des RTS classiques. Pour cette raison la plupart des MMORTS réduisent les simulations qui requièrent un degré de synchronisation élevé avec les serveurs comme la réduction des champs de vision ou la gestion des collisions. Par conséquent, les MMORTS ne représentent actuellement qu'une niche de joueurs réduite par rapport aux communautés gravitant autour des MMORPG (*Massively multiplayer online role-playing game*).

La majorité des innovations présentées ci-dessus ont été apportées par l'intermédiaire de logiciels propriétaires. Les projets de STR à code source ouvert qui peuvent servir de base à la conception d'un jeu sérieux dédié à l'apprentissage de la programmation intègrent nombre de ces innovations. À ce titre, la section suivante en présente quelques exemples.

23. `http://www.ogame.org/` accédé le 1 Juillet 2010.
24. `http://www.playsaga.com/` accédé le 1 Juillet 2010.
25. avatar : entité représentant un joueur dans un jeu vidéo

## 2.4 Exemples de jeux de STR à code source ouvert

De nombreux jeux de STR à code source ouvert existent et témoignent de l'intérêt que suscite ce type de jeu auprès des communautés de joueurs. Parmi ces projets, quelques-uns intègrent la 3D et un mode de jeu multijoueur à l'image de *Spring* [26], ORTS [27] et *WarZone 2100* [28].

### 2.4.1 Spring

Le projet *Spring*, inspiré du jeu commercial *Total Annihilation* (présenté dans l'historique de la section 2.2), a été amorcé par Stephan Johansson, membre du groupe *Swedish Yankspankers* [29] (SY). L'objectif initial de ce projet était de réutiliser le contenu du jeu commercial dans un nouveau moteur plus performant. Cet objectif ayant été atteint, le projet a évolué pour servir de base à de nouveaux jeux de RTS originaux.

*Spring* est donc un moteur de jeu de STR. Il prend en charge la simulation du jeu, la gestion des communications réseaux (architecture P2P), le rendu graphique et sonore et le chargement d'IA externes. Il dispose, en outre, d'un interpréteur Lua [30] qui permet aux utilisateurs de rassembler des scripts au sein d'une archive chargée et exécutée par le moteur. Grâce à cette fonctionnalité, chaque utilisateur est libre de créer son propre contenu et donc de concevoir avec *Spring* son propre jeu de STR.

*Spring* bénéficie d'une communauté de joueurs active. Elle participe à l'évolution du projet en proposant de nouveaux contenus (jeux, IA, cartes...) et à la fiabilisation du jeu en découvrant les bogues corrigés ensuite par le groupe de développeurs. Plusieurs jeux [31] sont actuellement disponibles sur *Spring* dont (sans exhaustivité) : *XTA* le premier jeu distribué par le groupe SY avec la première version de *Spring* ; *Balanced Annihilation* le jeu le plus joué par la communauté ; *Star Wars : Imperial Winter* basé sur l'univers de George Lucas ; et *Kernel Panic*.

Ce dernier (illustré par la figure 2.6) est un jeu de STR simplifié : il n'y a pas de gestion de ressources excepté le temps et l'espace ; toutes les unités sont gratuites ; « l'arbre des technologies » est peu important avec moins de dix unités par faction ; il utilise un rendu vec-

26. `http://springrts.com/` accédé le 15 Mai 2010.
27. `http://webdocs.cs.ualberta.ca/~mburo/orts/` accédé le 17 Mai 2010.
28. `http://wz2100.net/` accédé le 18 Mai 2010.
29. `http://www.clan-sy.com/` accédé le 15 Mai 2010.
30. `http://www.lua.org/` accédé le 15 Mai 2010.
31. `http://springrts.com/wiki/Games` accédé le 15 Mai 2010.

toriel original en adéquation avec l'univers du jeu. Ces caractéristiques mettent l'accent sur la stratégie et la tactique dans un jeu orienté action et accessible à tous les joueurs.

L'univers du jeu *Kernel Panic* se positionne au sein même d'un ordinateur où le joueur peut prendre le contrôle d'une des trois factions disponibles : les « Systèmes », les « Pirates » et les « Réseaux ». Chacune d'entre elles propose différentes unités comme par exemple le BIT, l'OCTET et l'ASSEMBLEUR chez les « Systèmes », le VIRUS, le BOGUE et le VER chez les « Pirates » et le PORT, le PAREFEU et le PAQUET chez les « Réseaux ». La figure 2.7 présente en détail la hiérarchie de création des unités des « Systèmes ». Ainsi le NOYAU (unité maîtresse de cette faction) peut générer des BITS, des OCTETS, des POINTEURS et des ASSEMBLEURS. Ce dernier peut à son tour générer des SOCKETS (qui pourront produire des BITS) et des TERMINAUX.

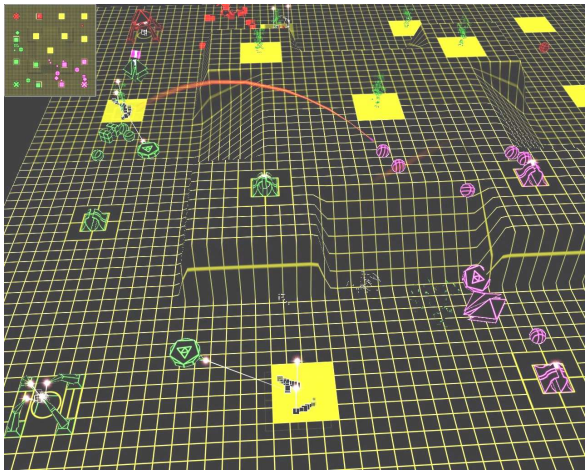


FIGURE 2.6 – Situation de jeu dans *Kernel Panic*.

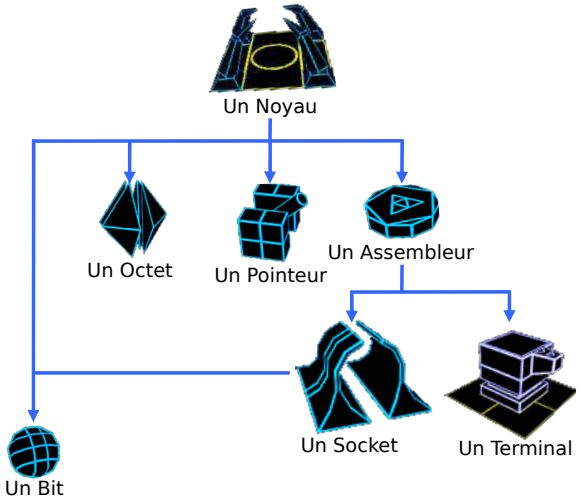



FIGURE 2.7 – Hiérarchie de création des unités des « Systèmes ».

### 2.4.2 ORTS

Ce jeu est principalement développé par Michael Buro et Timothy Furtak du département informatique de l'université des sciences d'Alberta (Canada). ORTS (*Open Real-Time Strategy*) est un environnement de programmation dédié à l'étude de problèmes d'IA temps réels (recherche de chemins, traitement d'informations incomplètes, planification de tâches...) dans un contexte de jeu de STR.

Buro et Furtak [BF05] notent que les jeux de STR commerciaux sont fermés et ne permettent pas aux chercheurs de connecter leurs modules d'IA aux jeux. Les IA des jeux de STR actuels souffrent d'un manque de planification et d'apprentissage, domaine dans lequel les joueurs sont toujours plus efficaces que les IA. Ils ont donc développé ORTS pour permettre aux programmeurs confirmés d'intégrer et de tester leurs IA dans un moteur de STR totalement libre. Ils précisent qu'ORTS est un moteur de jeu de STR (voir figure 2.8) qui permet aux utilisateurs de définir leurs propres jeux sous la forme de scripts qui décrivent les unités, les structures et leurs interactions.


FIGURE 2.8 – Situation de jeu dans ORTS

La composante originale d'ORTS concerne son architecture réseau destinée à éviter la tricherie lors des parties multi-utilisateur [Bur02]. L'architecture d'ORTS est basée sur un système centralisé client-serveur. Dans cette architecture, chaque client se connecte à un serveur et génère les actions des objets contrôlés par le joueur. Le serveur, quant à lui, envoie les informations visibles de chaque joueur vers son client respectif et reçoit les actions des objets pour les exécuter.

Plusieurs compétitions de programmation ont été organisées avec ORTS. Quatre situations de jeu étaient proposées : récolter un maximum de ressources en 10 minutes ; détruire autant d'adversaires que possible en 15 minutes ; détruire tous les bâtiments adverses en 20 minutes ; détruire autant d'adversaires que possible en 5 minutes. Les défis à relever lors de ces compé-

titions portent entre autres sur la recherche de chemins, la gestion de groupes d'unités, le combat à petite échelle, l'allocation de ressources et l'exploration.

### 2.4.3 WarZone 2100

*WarZone 2100* (illustré figure 2.9) est à l'origine un jeu commercial développé par *Pumpkin Studios* et publié par *Eidos Interactive*. Il a été distribué à partir de 1999 sur *Microsoft Windows* et la console *Playstation*. En décembre 2004, son code source et la majorité de ses données ont été transférés sous licence GNU GPL (*General Public License*). Les restrictions sur les séquences d'animation et les pistes sonores ont été ouvertes le 10 juin 2008. Comme *Spring*, *WarZone 2100* est basé sur une architecture P2P.

En raison de son origine commerciale, ce jeu bénéficie d'un scénario original. Ce dernier projette le joueur dans un avenir proche où les civilisations ont été anéanties suite à une série de frappes nucléaires. Alors que la plupart des survivants vivotent, un groupe de personnes tente de reconstruire une civilisation grâce aux technologies d'avant guerre.

FIGURE 2.9 – Situation de jeu dans *WarZone 2100*.


FIGURE 2.10 – Système de conception d'unités.

La particularité de ce jeu porte sur une utilisation originale de « l'arbre des technologies » à travers un système de conception d'unités (voir figure 2.10). Il donne l'opportunité au joueur de concevoir ses propres unités (composées d'un châssis, d'un système de propulsion et d'une tourelle) en fonction des technologies découvertes au cours des parties. Avec plus de 400

technologies différentes, ce jeu permet la mise en œuvre d'une grande variété d'unités et donc de stratégies potentielles.

À l'image de *Spring* et d'ORTS, *WarZone 2100* peut également être considéré comme un moteur pour créer des jeux. À l'aide d'un langage de script proche du C, appelé WZScript, chaque utilisateur est libre de modifier une grande partie du jeu original (telle que les unités, l'arbre de recherche, le *gameplay*...) et d'améliorer l'IA du jeu. Un éditeur de carte est également à la disposition des joueurs pour leur permettre de générer leurs propres zones de jeu.

# Conclusion

Les quelques exemples exposés ci-dessus présentent des similarités. Premièrement ils se définissent tous comme des moteurs de STR destinés à être améliorés ou à servir de base à la conception de nouveaux jeux. Cette philosophie de coopération et de partage induite par le logiciel libre permet à ces projets de proposer des outils fiables et fonctionnels.

La deuxième similitude porte sur la mise en œuvre d'interfaces pour la réalisation d'IA. Cette fonctionnalité ouvre la voie à de nouvelles et intéressantes méthodes d'interactions. Le premier exemple porte sur la réalisation de tournois d'IA autonomes. Un second exemple concerne la réalisation de systèmes hybrides où les joueurs humains conçoivent et utilisent des interfaces graphiques sophistiquées. Ces dernières permettent de déléguer certaines tâches à des modules d'IA afin d'augmenter les performances des joueurs dans le jeu. Ce dernier point est certainement la contribution majeure des jeux à code source ouvert à cette famille de jeu vidéo.

Ces quelques similarités constituent une base opportune à la conception d'un jeu sérieux centré sur la pratique de la programmation. Dans ce cadre, il convient de présenter l'informatique en tant que discipline scientifique.

# 3

# Formation en informatique

L'informatique influence d'une manière significative de nombreux domaines liés aux activités humaines. Dans nos sociétés, une grande partie de la population utilise l'informatique dans un cadre privé. D'un point de vue professionnel, de nombreux secteurs d'activité sont en demande de qualifications informatiques propres.

Pour répondre à cette diversité, le modèle de formation anglo-saxon structure l'informatique en cinq grandes disciplines. Ce modèle diffère du modèle français en ce qui concerne le découpage strict de la discipline. Toutefois, aucun rapport détaillé du modèle français n'existe à notre connaissance. Pour cette raison le modèle anglo-saxon est présenté en détail puis mis en perspective dans notre vision du modèle français.

Malgré l'omniprésence de l'informatique dans la société, les étudiants délaissent cette discipline et les structures de formation voient leurs effectifs stagner voire diminuer. Dans ce cadre, quelques exemples d'innovations pédagogiques sont présentés. Leur enjeu consiste à motiver les étudiants à intégrer et poursuivre des formations en informatique en vue d'éviter une pénurie d'informaticiens.

## 3.1 Analyse du modèle anglo-saxon

Depuis plus de quarante ans, quatre organisations majeures aux États-Unis proposent des directives pour les formations en informatique à l'université : l'ACM [32] (*Association for Comput-*

32. `http://www.acm.org/` consulté le 25 Juin 2010

*ing Machinery*), l'AIS [33] (*Association for Information Systems*), l'AITP [34] (*Association of Information Technology Professionals*) et l'IEEE-CS [35] (*Computer Society of the Institute for Electrical and Electronic Engineers*). Aujourd'hui, ces sociétés collaborent pour décrire la diversité des enseignements liés à l'informatique dans le modèle anglo-saxon.

### 3.1.1 Découpage et points communs des disciplines informatiques

Le rapport ACM/AIS/IEEE-CS [AAIC05] explicite le panorama des différentes branches de l'informatique à l'université. Ce rapport divise l'informatique en cinq disciplines majeures :

1. *Computer Engineering* (CE) dont l'application dominante concerne les systèmes embarqués (développement de dispositifs tant sur le plan logiciel que matériel) ;
2. *Computer Science* (CS) qui fournit des bases complètes pour permettre aux diplômés de s'adapter aux nouvelles technologies et aux nouvelles idées ;
3. *Information Systems* (IS) pour lesquels les jeunes diplômés doivent être performants en spécification, conception, et implémentation afin de déterminer les besoins d'une organisation ;
4. *Information Technology* (IT) qui a pour objectif de comprendre les systèmes informatiques et leurs logiciels et s'engage à résoudre tous les problèmes en rapport avec ces systèmes ;
5. *Software Engineering* (SE) dont l'objectif est de former un personnel capable de gérer des projets de logiciels importants, complexes et fiables.

Ces cinq disciplines couvrent l'ensemble des thèmes et domaines d'applications liés à l'informatique. Bien que chacune ait un objectif de formation propre, de nombreuses connaissances et aptitudes sont transversales à l'ensemble d'entre elles.

Les programmes d'études en informatique sont des combinaisons d'un ensemble de thèmes ayant des poids différents en fonction de la spécialité enseignée. Le rapport ACM/AIS/IEEE [AAIC05, chap. 3] définit 40 thèmes et indique leur poids dans chacune des cinq disciplines informatiques. Le poids est caractérisé par deux valeurs comprises entre 0 (faible) et 5 (élevée) qui représentent l'importance d'un thème pour le domaine. Ces deux valeurs indiquent respectivement l'exigence minimale et maximale attendue par la discipline relativement aux autres.

33. `http://home.aisnet.org/` consulté le 25 Juin 2010
34. `http://www.aitp.org/` consulté le 25 Juin 2010
35. `http://www.computer.org/` consulté le 25 Juin 2010

Le tableau 3.1 donne un exemple d'évaluation issue de ce rapport. Dans cet exemple, « Algorithmes et complexité » a une valeur minimale de 4 en CS ce qui en fait un thème important de cette discipline. A contrario, « Support technique » (avec une valeur maximale de 1) fait partie des thèmes peu ou pas abordé en CS. Dans un autre contexte, celui de IT, l'exigence de ces mêmes thèmes est inversée.

TABLE 3.1 – Exemple d'évaluation de thèmes en fonction des disciplines informatiques (Extrait de [AAIC05, p. 24]).

| **Thème** | **CS** | | **IT** | |
|---|---|---|---|---|
| | **min** | **max** | **min** | **max** |
| Algorithmes et complexité | 4 | 5 | 1 | 2 |
| Support technique | 0 | 1 | 5 | 5 |

Grâce à ces valeurs, il est alors possible de déterminer les thèmes fondamentaux qui requièrent la plus grande exigence pour l'ensemble des disciplines informatiques. Nous calculons l'exigence pour un thème particulier (noté « $E_t$ ») à l'aide de la formule 3.1 où « $eMin_{i,t}$ » représente l'exigence minimale attendue par la i$^{\text{ème}}$ discipline informatique pour le thème « $t$ ». Les quatre thèmes les plus fondamentaux parmi les 40 définis sont donc : « Les fondamentaux de la programmation » ($E_t = 3,4$) ; « L'interaction homme machine » ($E_t = 2,6$) ; « L'analyse des besoins techniques » ($E_t = 2,4$) ; « La conception de logiciels » ($E_t = 2,4$).

$$E_t = \frac{\sum_{i=1}^{nbDisciplines} eMin_{i,t}}{nbDisciplines} \tag{3.1}$$

Le rapport ACM/AIS/IEEE [AAIC05, chap. 3] définit également 60 aptitudes parmi 11 catégories et évalue leur poids pour chaque discipline informatique. Contrairement aux thèmes, l'exigence de chaque aptitude est estimée par une seule valeur comprise entre 0 et 5. En calculant la moyenne des exigences pour chaque aptitude (notée « $E_a$ » dans la formule 3.2 où « $e_{i,a}$ » représente l'exigence attendue par la i$^{\text{ème}}$ discipline informatique pour l'aptitude « $a$ »), nous pouvons déterminer les trois premières aptitudes fondamentales d'un informaticien parmi les 60 définies : « Réaliser des programmes simples » ($E_a = 4,2$) ; « Créer une interface utilisateur » ($E_a = 4$) ; « Maîtriser un traitement de texte » ($E_a = 3,8$).

$$E_a = \frac{\sum_{i=1}^{nbDisciplines} e_{i,a}}{nbDisciplines} \quad (3.2)$$

L'analyse de ce rapport a permis d'identifier les deux composantes essentielles et communes aux cinq disciplines informatiques : « **acquérir les fondamentaux de la programmation** » et être capable de « **réaliser des programmes simples** ».

D'après le TLFi, le terme « informatique » caractérise « *la science du traitement rationnel, notamment par machines automatiques, de l'information considérée comme le support des connaissances humaines et des communications dans les domaines technique, économique et social* ». Ce terme a été inventé en France, en 1962, par Philippe Dreyfus. À cette même époque, aux États-Unis, les premières formations émergent et sont de trois sortes [AAIC05, chap. 2] : *Computer Science* (CS), *electrical engineering* (EE) et *Information Systems* (IS). Les étudiants qui souhaitaient devenir experts dans le développement de logiciels ou acquérir les compétences théoriques sur l'utilisation des ordinateurs suivaient des formations en CS. Ceux qui souhaitaient travailler sur le matériel suivaient des formations en EE. Enfin, ceux qui souhaitaient utiliser le matériel et le logiciel pour résoudre des problèmes liés aux entreprises suivaient des formations en IS. Dans ce contexte l'informatique telle qu'elle est définie en France fait référence au CS. C'est au cours des années 90 que sont apparues aux États-Unis les nouvelles disciplines comme le *Computer Engineering* (CE) le *Software Engineering* (SE) et l'*Information Technology* (IT). Cette décomposition n'ayant pas été réalisée dans le modèle français, l'informatique en France regroupe bien entendu le CS mais aussi tout ou partie du CE, du SE, de l'IT et de L'IS. Le CS est donc la discipline du modèle anglo-saxon la plus proche historiquement de l'informatique et mérite donc d'être présentée en détail.

### 3.1.2 Analyse du *Computer Science*

Cette discipline est présentée de manière plus détaillée dans un second rapport de l'ACM/-IEEE [AIC01]. Les parties suivantes effectuent une synthèse de ce rapport à travers la présentation de l'architecture des enseignements, le détail des différentes approches envisageables et le positionnement de l'apprentissage de la programmation.

### Architecture des enseignements

Le rapport de l'ACM/IEEE [AIC01] structure les connaissances en trois niveaux : les **domaines** qui représentent les différents champs disciplinaires ; les **unités** qui constituent les modules thématiques de chaque domaine ; et les **thèmes** qui composent les cours de chaque unité.

Les enseignements, à proprement parler, sont divisés en trois catégories en rapport avec le niveau auquel ils se produisent. Les enseignements désignés comme **introductifs** sont typiquement les cours dispensés en première année universitaire. Les enseignements **intermédiaires** se rattachent aux cours de deuxième et troisième années et fournissent des bases requises pour aborder les enseignements **avancés** des dernières années.

Chaque formation est définie par un cœur et un ensemble d'unités facultatives. Le cœur se compose d'unités d'enseignements requises pour tous les étudiants en CS. Toutefois, le cœur à lui seul ne suffit pas à l'instruction complète, les unités facultatives ont donc pour objectif de parfaire la formation. Pour offrir une grande flexibilité sur la mise en œuvre de ces enseignements, le rapport propose plusieurs mises en œuvre des différentes catégories. À titre d'illustration, les enseignements introductifs peuvent être proposés sous six approches différentes : impérative, orientée objet, fonctionnelle, étendue, algorithmique et matérielle. Nous noterons que pour chaque approche, il n'est fait aucune recommandation sur un langage de programmation à utiliser.

### Détail des six approches des enseignements introductifs

**Approche impérative :** Cette approche est la plus répandue, elle aborde l'introduction à la programmation via un style de programmation impératif. Il est important de noter que les premiers enseignements de cette approche peuvent être réalisés à l'aide d'un langage orienté objet (OO) pour illustrer les exemples et exercices de programmation. La différence entre cette approche et le modèle OO porte sur l'accentuation et l'ordonnancement des premiers enseignements. Même si un langage de programmation OO est utilisé, les premiers enseignements se concentrent sur les aspects impératifs du langage : expressions, structures de contrôle, procédures et fonctions, etc. Les techniques de conception OO sont alors reportées dans les enseignements futurs. Adopter l'approche impérative signifie que les étudiants seront moins exposés aux techniques de la programmation OO que s'ils avaient suivit l'approche du même nom. Le fait de positionner la conception OO au second plan est souvent perçu comme une faiblesse de cette approche.

**Approche orientée objet :** Cette approche se concentre également sur la programmation, elle met l'accent sur les principes de la conception OO dès le début des enseignements. Les premiers cours portent sur les notions d'objets et d'héritage afin de familiariser très tôt les étudiants à ces concepts. Après avoir expérimenté ces notions dans un contexte de programmes interactifs simples, les enseignements s'orientent vers l'introduction des traditionnelles structures de contrôle mais toujours dans un contexte lié à la conception OO. Les enseignements futurs abordent ensuite l'algorithmique, les structures de données fondamentales et les problèmes de l'ingénierie du logiciel. Une critique récurrente sur cette approche porte sur la complexité des langages utilisés dans les travaux pratiques, tels que le Java ou le C++. Par conséquent, les enseignants doivent veiller à introduire ces langages issus du monde professionnel d'une manière à limiter leur complexité pour ne pas submerger des étudiants novices en la matière.

**Approche fonctionnelle :** Cette approche se caractérise par l'utilisation d'un langage fonctionnel qui présente les avantages suivants : ce paradigme de programmation est peu connu des étudiants ce qui rend les promotions plus homogènes ; la syntaxe minimale des langages fonctionnels permet de concentrer le cours sur les notions fondamentales ; plusieurs concepts importants, comme la récursivité, les structures de données liées et les fonctions sont exprimées très naturellement et peuvent être abordées plus tôt dans la formation. Une critique, souvent mise en évidence dans cette approche, porte sur la nécessité de demander aux étudiants un raisonnement plus abstrait que pour les langages de programmation traditionnels et ceci très tôt dans la formation. Alors que cette compétence est précieuse et nécessaire à la formation d'un informaticien, la placer trop tôt peut décourager les étudiants non familiarisés avec ce type de raisonnement. Enfin, pour couvrir l'ensemble des compétences requises dans un enseignement introductif, les notions liées à la conception et à la programmation OO devront être abordées dans la deuxième partie du cours.

**Approche étendue :** D'après cette approche, aborder l'enseignement de l'informatique par la programmation donne une vision limitée de la discipline aux étudiants. En effet, l'informatique est un domaine en constante évolution qui inclut de nombreuses activités au delà de la simple programmation. Les enseignements qui la privilégient peinent à sensibiliser les étudiants aux autres domaines et styles de pensée qui font de l'informatique un tout. Pour fournir une vision plus holistique de la discipline, les premiers enseignements présentent l'informatique relativement à l'environnement dans lequel elle se manifeste. L'avantage principal de cette approche

est de fournir aux étudiants une vision globale de la discipline pour leur permettre de déterminer s'ils souhaitent l'étudier en profondeur. Le principal inconvénient de cette approche reste l'ajout de cours supplémentaires qui diffèrent dans le temps les enseignements introductifs classiques.

**Approche algorithmique :** Dans cette approche, les concepts basiques de l'informatique sont introduits à l'aide de pseudocodes au lieu de langages exécutables. En introduisant les concepts basiques d'algorithmique sans langage particulier, cette approche minimise les préoccupations liées aux détails syntaxiques de la programmation. Elle demande aux étudiants de raisonner et d'expliquer la conception de leurs algorithmes en les déroulant manuellement. Elle leur permet de travailler sur des ensembles de données et des structures de contrôle sans affronter les particularités inévitablement liées aux langages de programmation. Lorsque les étudiants ont des bases solides en algorithmique ainsi que sur la manipulation des données et des structures de contrôle en pseudocode, ils abordent un langage plus conventionnel. En raison de leur expérience plus poussée en algorithmique, les étudiants progressent plus rapidement dans leur maîtrise du langage. Ils peuvent se concentrer sur des notions de programmation efficaces ou le débogage. Par ailleurs, les étudiants peuvent se sentir frustrés de ne pouvoir tester leurs algorithmes dans un contexte réel. Les premiers enseignements étant centrés sur l'algorithmique, la deuxième partie devra aborder les techniques de conception et de programmation OO.

**Approche matérielle :** Cette approche enseigne les bases de l'informatique au niveau de la machine puis intègre des concepts de plus en plus abstraits. Tout d'abord, elle aborde les circuits avec l'utilisation de registres et d'unités arithmétiques simples, puis les contextualisent dans une machine de von Neumann. La programmation est alors abordée avec un langage de haut niveau de type OO. Alors que cette approche fonctionne bien avec les étudiants qui préfèrent comprendre le processus de calcul dans tous ces détails, elle est moins efficace face au développement croissant du logiciel et des machines virtuelles qui tentent de séparer le processus de programmation du matériel sous-jacent. Cette approche est donc particulièrement bien adaptée pour la discipline CE où une exposition précoce au matériel est essentielle.

#### Importance des fondamentaux de la programmation

La section 3.1.1 a permis d'identifier « les fondamentaux de la programmation » comme une connaissance primordiale en informatique. Qu'en est-il pour le *Computer Science ?*

En CS, cette connaissance est structurée sous l'appellation « Fondamentaux de la Programmation (FP) » divisés en cinq unités d'enseignements : « Fondamentaux de la construction de programmes (FP1) » ; « Algorithmes et résolution de problèmes (FP2) » ; « Fondamentaux sur les structures de données (FP3) » ; « Récursivité (FP4) » ; et « Programmation évènementielle (FP5) ». Les thèmes abordés dans ces différentes unités portent sur la manipulation de variables, les structures de contrôle, l'utilisation de débogueurs, les tableaux, les pointeurs et les références, la propagation d'évènements, etc.

Quelle que soit l'approche (impérative, fonctionnelle, etc.), les FP représentent une part importante du cœur des différentes formations. Toutefois, la proportion des FP vis-à-vis des autres unités d'enseignements varie en fonction du modèle de mise en œuvre. Historiquement, la majorité des établissements utilisent un modèle en deux semestres pour les enseignements introductifs. Mais alors que le contenu des leçons a évolué au cours du temps pour répondre aux changements des technologies et des approches pédagogiques, la longueur des enseignements est restée la même. Le rapport de l'ACM/IEEE [AIC01] propose donc pour certaines approches (impérative, orientée objet et étendue) un modèle en trois semestres de manière à fournir une vision plus large et plus complète de l'informatique. Le choix d'une approche et d'un modèle de mise en œuvre est à la charge de l'équipe pédagogique. En effet, parmi l'ensemble des combinaisons possibles, une seule approche de l'un des deux modèles est nécessaire pour assurer l'ensemble des enseignements informatiques des enseignements introductifs.

Outre la présentation de l'architecture modulaire des enseignements introductifs, la figure 3.1 indique l'importance des fondamentaux de la programmation dans le cœur des différentes formations (39% de l'approche impérative du modèle en trois semestres, 37% de l'approche impérative du modèle en deux semestres, 40% de l'approche algorithmique, etc.). Dans leur majorité, les unités d'enseignements traitant des FP font parties du cœur de toutes les approches et sont abordées lors des premières années (en moyenne : 30,7% des enseignements introductifs, 5,85% des enseignements intermédiaires et 0% des enseignements avancés). Cette répartition positionne les FP comme l'un des thèmes centraux des enseignements introductifs indispensables aux étudiants pour aborder la suite de leurs études en CS. Les « Autres enseignements » sont complémentaires des enseignements informatiques et doivent être associés à l'approche choisie par l'équipe pédagogique.

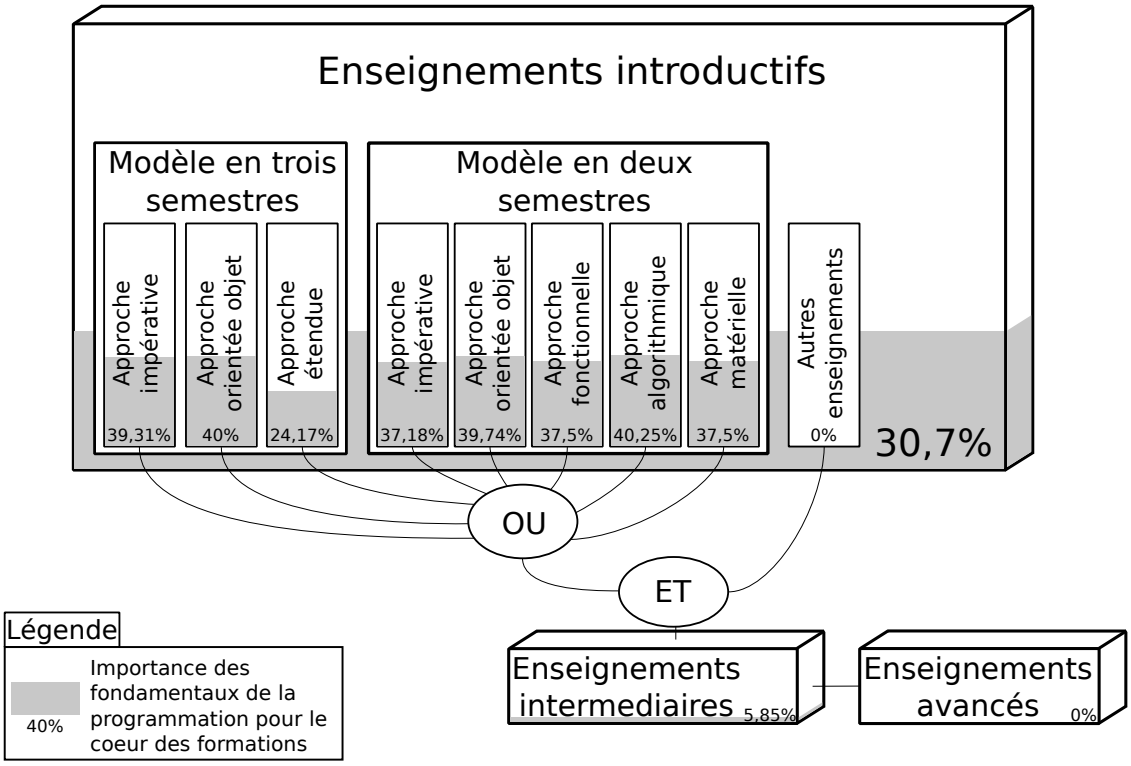



FIGURE 3.1 – Architecture modulaire des enseignements introductifs et importance des fondamentaux de la programmation

### 3.1.3 Comparaison avec le modèle français

Comme il n'existe pas à notre connaissance de rapport détaillé sur l'enseignement de l'informatique en France, nous usons de notre expérience d'enseignant pour comparer le modèle anglo-saxon à notre vision du modèle français.

La principale différence identifiée porte sur la décomposition en disciplines qui ne peut être reportée directement dans le modèle français. En effet, en France l'informatique en tant que discipline scientifique regroupe le CS et tout ou partie des autres disciplines du modèle anglo-saxon. Toujours est-il que l'acquisition des fondamentaux de la programmation et la capacité à réaliser des programmes simples sont deux composantes essentielles dans l'enseignement de l'informatique en France. Pour illustrer ces propos, le PPN (Programme Pédagogique National) du DUT Informatique [Min05] définit comme objectif général de la formation de « *former les étudiants à être capable de participer à la conception, la réalisation et la mise en œuvre de systèmes informatiques correspondant aux besoins des utilisateurs* ». Dans ce cadre, le module

« Algorithmique et Programmation » fait référence aux FP du modèle anglo-saxon et représente une part importante de l'unité d'enseignement « Informatique » avec 41% des enseignements de première année et 24% des enseignements de deuxième année. Les fiches métiers ROME (Répertoire Opérationnel des Métiers et des Emplois) créées par le Pole emploi [36] illustrent également les activités et compétences attendues en France pour un métier particulier. Concernant les professions relatives aux « études et développement informatique » (fiche ROME M1805) « la conception et le développement de programmes et applications informatiques » et « l'algorithmique » font respectivement parties des six activités et huit compétences basiques de cette fiche.

Les six approches de l'enseignement introductif, quant à elles, peuvent être parfaitement intégrées au modèle français. Les approches impératives, algorithmiques, OO et fonctionnelles sont cependant les plus répandues. Pour illustrer ce propos, la nouvelle habilitation de l'Université Paul Sabatier évaluée par l'AERES (Agence d'Évaluation de la Recherche et de l'Enseignement Supérieur) change son offre d'enseignement d'une approche fonctionnelle en une approche impérative pour les années 2011 à 2014. Un autre exemple porte sur l'intégration de l'algorithmique au programme de Mathématiques de seconde générale et technologique en lycée. Cette réforme est entrée en application à la rentrée de l'année scolaire 2009-2010 [Bul09].

Malgré les différences de structuration de la discipline entre le modèle anglo-saxon et le modèle français, le savoir enseigné et les méthodes d'enseignement restent les mêmes.

## 3.2 Analyse de l'attractivité de la discipline informatique

L'enseignement de l'informatique est né avec le développement des ordinateurs centraux aux États-Unis et en Europe à la fin des années 40. L'avènement des microprocesseurs à la fin des années 70 a relancé la discipline mais depuis quelques années, le nombre d'étudiants inscrit en informatique dans des pays où cet enseignement était jusqu'alors en expansion commence à décliner. De nombreux travaux relatent ce phénomène comme ceux de Crenshaw *et al.* [CCMT08] et de Kelleher [Kel06]. Le rapport de l'ACM/AIS/IEEE [AAIC05, p. 39] souligne également cette tendance : « *les universités affirment que régulièrement 50% ou plus de leurs étudiants ayant initialement choisi des études en informatique décident rapidement d'abandonner* ». La dernière mise à jour de ce rapport consacrée au *Computer Science* [AIC08]

36. `http://www.pole-emploi.fr/accueil/` consulté le 28 Juin 2010

dédie un chapitre entier à ce qu'elle nomme « *la crise de l'informatique* » dans l'enseignement. Notre université subit le même phénomène avec une baisse de 30% d'étudiants inscrits en deuxième année de licence informatique sur les sept dernières années (148 inscrits en 2003, 104 inscrits en 2009).

La raison de ce déclin reste encore floue, toutefois « *les chercheurs en pédagogie s'accordent à dire que les étudiants novices en informatique trouvent souvent la discipline théorique, technique, ou ennuyeuse* » Stamm [Sta07]. En effet, l'environnement informatique utilisé tous les jours par les étudiants, pour jouer ou pour dialoguer par exemple, est très différent des dispositifs déployés dans les enseignements. Ainsi, les jeunes perçoivent l'importance de l'informatique dans leur quotidien et dans les activités professionnelles comme la production de musique et de film, le développement de sites Web ou de jeux vidéo, mais ont des difficultés à faire le lien avec les enseignements dispensés en informatique.

D'autre part, se former en informatique est une activité complexe, en particulier pour les novices, comme en témoigne l'apprentissage des fondamentaux de la programmation. Dans les unités d'enseignements qui composent ce domaine, les étudiants se trouvent confrontés à plusieurs obstacles épistémologiques comme la manipulation des boucles (Ginat [Gin04] et Soloway *et al.* [SBE83]), des conditions, ou l'assemblage de programmes à base de composants. Les structures de contrôle et les algorithmes posent souvent problème de par leur abstraction et leur nature dynamique (Seppälä *et al.* [SMK06]. Pour apprendre à programmer les étudiants doivent donc connaître les concepts et connaissances liées aux unités d'enseignement, mais pour les maîtriser, ils doivent pratiquer la programmation.

Pour tenter d'apporter une solution à la « *la crise de l'informatique* » dans l'enseignement, il convient de remotiver les étudiants à travers de nouvelles innovations pédagogiques. Ces innovations ne doivent pas dénaturer le contenu fondamental de la discipline ou réduire son niveau académique. En effet, les étudiants doivent acquérir les prérequis nécessaires à leur poursuite d'étude dans le cadre des enseignements intermédiaires et avancés.

### 3.2.1 Motivation et performances

Que signifie le terme « motivation » et quelle peut être son influence sur les performances des étudiants et donc sur leur apprentissage ? Viau [Via97] définit la motivation en contexte scolaire comme suit : « *...état dynamique qui a son origine dans les perceptions qu'un élève a de lui-même et de son environnement et qui l'incite à choisir une activité, à s'y engager et à per-*

*sévérer dans son accomplissement afin d'atteindre un but* ». Dans son modèle, la performance d'un élève à réaliser une activité correspond aux résultats observables de l'apprentissage et constitue une conséquence de la motivation (elle en est un indicateur). Agir sur la motivation permet indirectement de faire évoluer les performances des étudiants. Dans ce contexte, la motivation est caractérisée par des déterminants et des indicateurs.

Les déterminants de la motivation concernent :

- **la perception de la valeur d'une activité**, c'est un jugement que l'étudiant porte sur l'utilité d'une activité en vue d'atteindre des buts qu'il poursuit. La détermination de buts à court, à moyen et à long terme est à la base de la perception de la valeur d'une activité (concept de perspective future). Ces buts peuvent être classés en deux grandes catégories :
  1. les buts sociaux concernent la relation qu'un élève établit avec les autres élèves et avec l'enseignant ;
  2. Les buts scolaires qui ont trait à l'apprentissage et à ses compétences dont les buts d'apprentissage (ceux que nous poursuivons lorsque nous accomplissons une activité pour acquérir des connaissances - motivation intrinsèque) et les buts de performance (ceux que nous poursuivons lorsque nous voulons réussir une activité pour que les autres nous estiment et nous reconnaissent ou encore pour obtenir une récompense, des félicitations, etc. - motivation extrinsèque).
- **la perception de sa compétence à accomplir une activité**, c'est une perception de soi par laquelle la personne, avant d'entreprendre une activité qui comporte un degré élevé d'incertitude quant à sa réussite, évalue ses capacités à l'accomplir de manière adéquate. Selon Bandura [Ban86], cette perception provient de quatre sources :
  1. les performances antérieures qui correspondent aux succès et échecs passés d'un étudiant ;
  2. l'observation d'autres personnes à exécuter une activité. Observer un pair influence davantage la perception qu'un étudiant à de sa compétence que d'observer un professeur ;
  3. la persuasion dont le but est de convaincre un élève de ses capacités à accomplir une activité ;
  4. les réactions physiologiques et émotives sont également une source de la perception qu'un élève à de sa compétence.

– **la perception de la contrôlabilité d'une activité**, c'est la perception qu'un élève a du degré de contrôle qu'il possède sur le déroulement et les conséquences d'une activité qu'on lui propose de faire. L'impuissance apprise est probablement la forme la plus extrême de perception d'incontrôlabilité qu'un élève puisse ressentir. Weiner [Wei92] a classé les causes de la contrôlabilité selon trois dimensions :

  1. le lieu de la cause permet de faire la distinction entre les causes internes à l'élève et les causes externes ;
  2. la stabilité de la cause. Une cause est dite stable lorsqu'elle a un caractère permanent aux yeux de l'étudiant. A l'opposé, une cause qui, comme l'effort, est susceptible de fluctuer régulièrement est dite modifiable.
  3. le contrôle de la cause. Une cause est dite contrôlable lorsqu'un élève perçoit qu'il aurait pu l'éviter s'il avait voulu ; par contre, elle est dite incontrôlable lorsqu'il perçoit qu'il n'avait aucun pouvoir sur elle.

Les indicateurs de la motivation sont caractérisés par :

– **le choix d'entreprendre une activité**, un élève motivé choisit d'entreprendre une activité d'apprentissage tandis qu'un élève démotivé a tendance à l'éviter ;
– **la persévérance**, mesurée en calculant le temps que l'élève consacre à des activités comme la prise de notes, l'accomplissement d'exercices, la compréhension de ses erreurs, l'étude de manuels, etc. ;
– **l'engagement cognitif**, défini comme l'utilisation par l'élève de stratégies d'apprentissage et de stratégies d'autorégulation lorsqu'il accomplit une activité. Les stratégies d'apprentissage sont principalement de trois types :

  1. les stratégies de mémorisation (exemple : copier ses connaissances dans un cahier de note) ;
  2. les stratégies d'organisation (exemple : faire des tableaux, des schémas) ;
  3. les stratégies d'élaboration (exemple : résumer un texte, définir les concepts dans ces propres mots).

  Les stratégies d'autorégulation sont des stratégies cognitives qui sont utilisées par l'élève ; en ce sens, elles ne sont pas observables. Zimmerman [Zim86] les classe en trois catégories :

1. les stratégies métacognitives correspondent à la conscience qu'une personne à de son fonctionnement cognitif et des stratégies qu'elle utilise pour réguler sa façon de travailler intellectuellement (planification, monitoring et autoévaluation) ;
2. les stratégies de gestion ont trait à l'organisation de l'apprentissage. L'organisation du travail dans le temps, le lieu d'apprentissage, et les ressources humaines et matérielles requises caractérisent cette organisation de l'apprentissage ;
3. les stratégies motivationnelles sont utilisées pour augmenter ou conserver la motivation à accomplir une activité.

– **la performance**, correspond aux résultats observables de l'apprentissage. C'est une conséquence de la motivation et c'est ce qui en fait un indicateur. Toutefois, ne pas oublier la relation inverse, c'est-à-dire l'influence qu'exerce la performance sur la motivation de l'étudiant. La performance n'est pas seulement une conséquence de la motivation, elle en est également une source.

Intervenir sur une seule composante motivationnelle de ce modèle peut influencer positivement toutes les autres et indirectement améliorer les performances des étudiants. L'innovation pédagogique s'inscrit dans cette dynamique.

### 3.2.2 Innovations pédagogiques

La recherche dans le domaine de l'enseignement de l'informatique consacre une part importante de ces travaux aux problèmes du recrutement et du maintien des étudiants dans les formations informatiques [FPC+04]. De nombreuses réponses sont donc proposées dans la littérature pour aider les étudiants à dépasser les difficultés présentées en introduction de cette section. Les propositions sont généralement de deux types : la première consiste à faire évoluer l'environnement de travail des étudiants, la seconde consiste à travailler les exercices proposés aux étudiants.

#### Construction d'environnements de programmation pour novices

De nombreux environnements de programmation pour novices ont été développés. La plupart utilisent des langages de programmation à base de blocs. Dans ce contexte chaque bloc (ou brique) représente un élément du langage comme une structure de contrôle, un opérateur, une variable, une fonction, etc. Ces dernières peuvent être combinées par « glisser-déposer » en

vue de construire un programme informatique. La couleur ou la forme des briques symbolise la grammaire du langage et aident les programmeurs à les combiner de manière correcte. Cette métaphore de programmation permet donc aux étudiants de faire abstraction de la syntaxe, souvent source d'erreurs, pour directement s'intéresser à la résolution de problèmes.

**Scratch** [37] [MBK+04] (voir figure 3.2) est un environnement de programmation utilisable par les enfants pour créer leurs propres histoires animées, jeux vidéo ou créations interactives et les partager sur le Web.

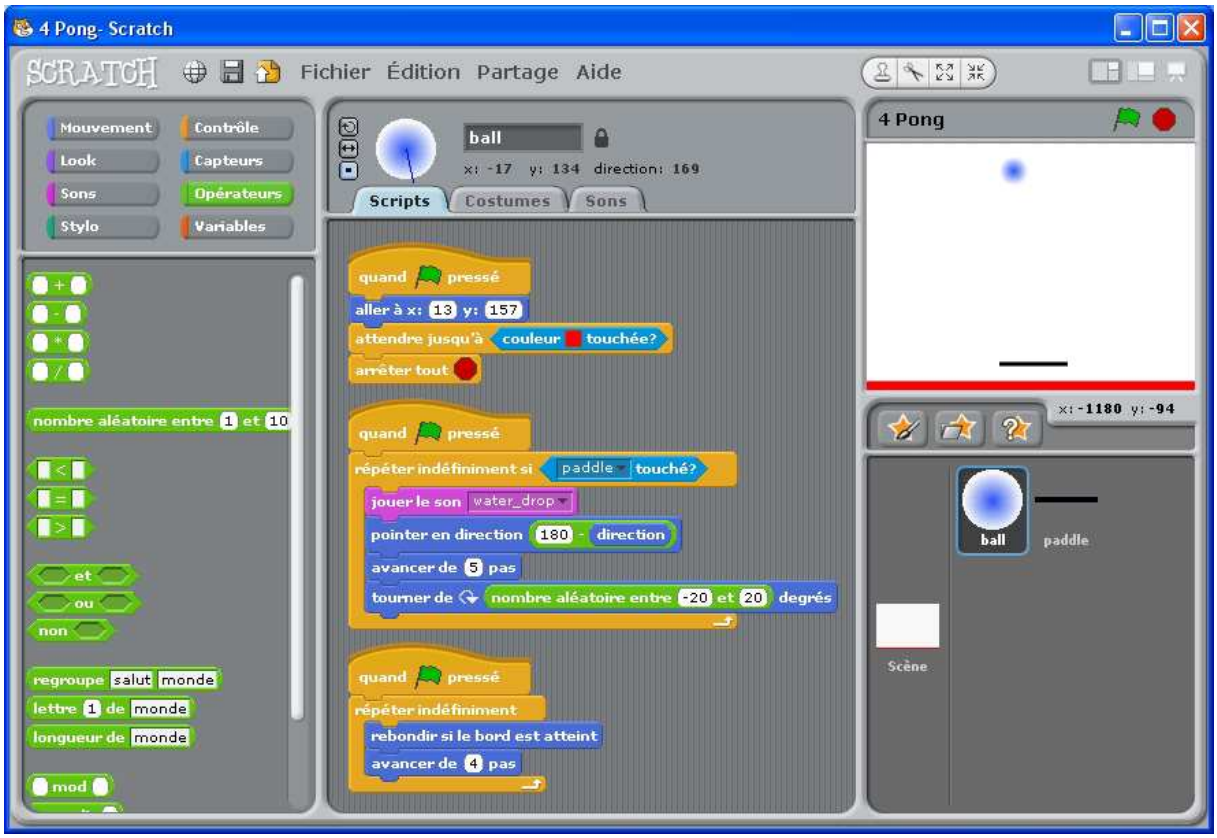



FIGURE 3.2 – Scratch

Alors que les enfants créent et échangent des projets Scratch, ils apprennent les notions importantes de mathématiques, de calcul et de programmation tout en travaillant de manière collaborative leur créativité et leur raisonnement. Scratch a donc été conçu dans un esprit d'enseignement et d'apprentissage. Il peut être utilisé dans de nombreux contextes différents : dans

37. `http://scratch.mit.edu/` consulté le 29 Juin 2010

l'enseignement (école, collège et lycée), dans les musées, les centres de loisir et à la maison. Cette application est notamment mentionnée comme l'un des logiciels utilisable comme support à l'enseignement de l'algorithmique en seconde générale et technologique au lycée [Edu09]. Il a été conçu spécialement pour les jeunes de 8 à 16 ans mais peut être également utilisé par les enfants avec l'aide de leurs parents et à l'université dans les enseignements d'introduction à la programmation. Autour de cette application gravite une communauté d'enseignants qui partagent en ligne leurs expériences et leurs ressources. L'interface graphique de Scratch est notamment traduite en une cinquantaine de langues différentes. Scratch est développé par le *Lifelong Kindergarten Group* du *MIT Media Lab*, avec le support financier du *National Science Foundation*, de *Microsoft*, de l'*Intel Foundation*, de *Nokia*, d'*Iomega* et du *MIT Media Lab research consortia*.

Les éléments clés intégrés à Scratch sont un système de programmation à base de blocs, un outil de manipulation de médias et un système de partage et de collaboration. Le langage de bloc utilisé par Scratch permet de manipuler les concepts de séquence, d'itération, d'instruction conditionnelle, de variable et de liste. Il fournit également les briques utiles à la programmation évènementielle et parallèle et à la gestion des périphériques d'entrée tels que le clavier et la souris. En revanche la version actuelle de Scratch ne fournit pas de briques pour créer des procédures et des fonctions, pour aborder la programmation OO, pour gérer les exceptions et pour manipuler les fichiers. Par conséquent les concepts liés au passage de paramètres, à la récursivité, à la définition de classes et à l'héritage ne peuvent pas être traités avec cet environnement de programmation.

Scratch définit trois types de blocs : les « blocs de pile », les « chapeaux » et les « rapporteurs ». Les « blocs de pile » sont caractérisés par une bosse en dessous et une encoche en dessus à l'image du bloc suivant : 

. Ces blocs peuvent donc être emboîtés les uns dans les autres pour former des piles et acceptent, pour certains, des paramètres en entrée. Les « blocs de pile » peuvent être comparés à des procédures. Les « chapeaux » sont des blocs placés en haut de pile. Ils attendent des évènements particuliers pour lancer l'exécution de la pile sous-jacente. Les « rapporteurs » sont conçus pour s'insérer dans les zones d'entrée des autres blocs. Ils peuvent rapporter trois types de données : un nombre, une chaîne de caractère ou un booléen. Les « rapporteurs » peuvent être comparés à des fonctions.

**StarLogo TNG (*The Next Generation*)** [38] [KY05] (voir figure 3.3) est le successeur de StarLogo lui même héritier du langage Logo (de Papert).


FIGURE 3.3 – StarLogo The Next Generation

Alors que cette version conserve la philosophie du premier StarLogo (un outil utile pour créer et comprendre la simulation de systèmes complexes), il apporte deux innovations principales : un système de programmation à base de blocs pour attirer un public plus jeune à pratiquer la programmation et un environnement 3D afin d'augmenter la richesse des jeux et des modèles de simulation réalisés. StarLogo TNG est développé par le *MIT Scheller Teacher Education Program*.

La structuration des blocs dans StarLogo TNG s'articule autour d'une quinzaine de palettes. Certaines donnent accès à la gestion des conditions initiales et à l'exécution du programme, au contrôle du mouvement des agents, à la manipulation du son et de l'environnement 3D (terrain et agents), aux périphériques de contrôle (clavier et joystick) et à la gestion du point de vue. D'autres permettent d'accéder aux structures de contrôle, aux opérateurs logiques et à la manipulation des variables et des listes. Dans StarLogo TNG, les types manipulés sont les nombres, les booléens et les chaînes de caractères. Des palettes spécifiques fournissent les blocks

38. `http://education.mit.edu/drupal/starlogo-tng` consulté le 29 Juin 2010

nécessaires aux calculs mathématiques et à la manipulation des chaînes de caractères. Enfin StarLogo TNG autorise la création et l'utilisation de procédures avec passage de paramètres mais ne fournit pas de support pour aborder les concepts de la programmation orientée objet.

Enfin, StarLogo TNG bénéficie d'une communauté active qui propose à travers le site Web [39] une série de didacticiels qui présentent l'outil et ses performances notamment en termes de simulation et de système multi-agents. Des supports de cours complets sont également à la disposition des enseignants et abordent des thèmes comme la création d'un jeu vidéo ou la simulation de l'activité des termites, d'un feu de forêt et d'une épidémie entre autres. Chaque cours propose une ingénierie didactique avec pour la plupart : supports de cours, ressources nécessaires aux étudiants, corrections, feuilles d'exercices et évaluations.

**Alice** [40] (voir figure 3.4) est un environnement de programmation 3D dédié à la création de jeux vidéo, de cinématiques à partager sur le Web ou d'animations en vue de raconter des histoires. C'est également un outil d'enseignement qui permet aux étudiants d'apprendre les fondamentaux de la programmation avec une approche OO [CDP03]. Dans Alice, des objets 3D (personnages, animaux et véhicules) peuplent le monde virtuel et les étudiants créent des programmes pour les animer. La programmation est réalisée à l'aide d'un système de blocs qui permet une programmation OO et une forme simplifiée de programmation parallèle. Ce système est donc naturellement composé d'un ensemble de briques pour manipuler les structures de contrôle, accéder aux opérateurs propres au langage et gérer les évènements utilisables dans l'environnement. Chaque objet du monde virtuel est également composé d'un ensemble de briques qui représentent les attributs, méthodes et fonctions encapsulés dans l'objet.

Deux versions d'Alice sont actuellement disponibles. La première version, *Alice 2.2*, est dédiée à un public lycéen voire premières années universitaires. La deuxième version, *Storytelling Alice*, développée par Caitlin Kelleher [KPK07] a été conçu pour un public plus jeune (collège) et féminin. Les expériences réalisées avec ces outils semblent montrer une réelle capacité à motiver les élèves à poursuivre une formation en informatique.

À l'image de Scratch et de StarLogo TNG, Alice bénéficie d'une communauté d'utilisateurs (étudiants et enseignants) fédérée autour des forums et ressources accessibles via le site Web. Actuellement plus de 2000 établissements de formation attestent avoir déjà utilisé Alice dans leurs enseignements.

39. `http://education.mit.edu/drupal/starlogo-tng/learn` consulté le 29 Juin 2010
40. `http://www.alice.org/` consulté le 30 Juin 2010

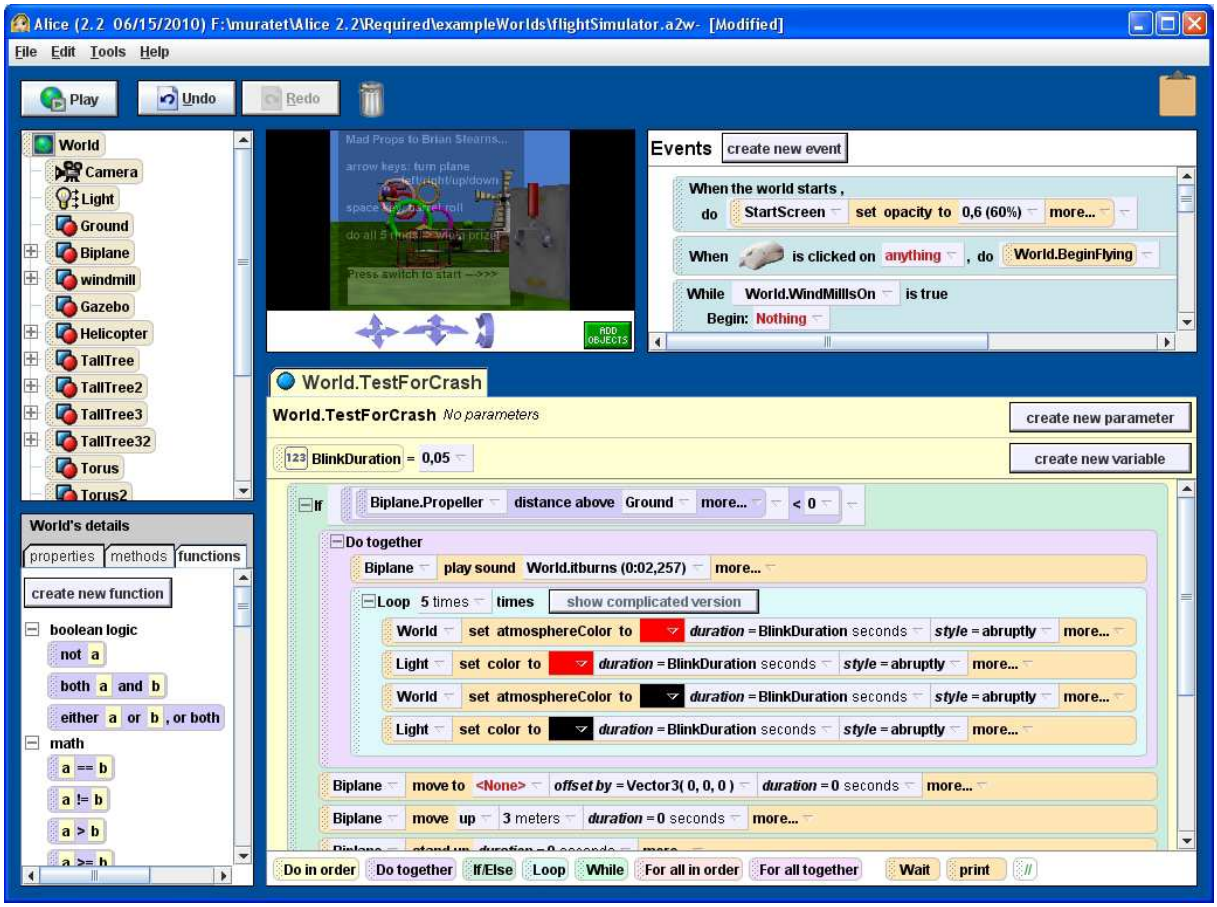



FIGURE 3.4 – Alice

**Kodu** [41] est développé par *Microsoft Research* sur Windows et Xbox 360, cet environnement de programmation se veut accessible aux enfants et divertissant pour tout public. Cet outil est dédié spécialement à la création de jeux vidéo et fournit un environnement complet de conception, de réalisation et de test des jeux produits. L'interface de programmation est le cœur de cet outil. Elle intègre un langage simple entièrement à base d'icônes. Un programme est structuré en pages qui sont décomposées en règles, elles-mêmes divisées en conditions et actions. Les conditions sont toutes évaluées simultanément. Le langage de Kodu étant conçu spécialement pour la création de jeux, il fournit des primitives adaptées. Les programmes sont exprimés sous la forme de termes physiques utilisant les concepts de vision, d'audition, et de temps pour contrôler le comportement des personnages. Contrairement aux langages de programmation

41. `http://fuse.microsoft.com/projects-kodu.html` consulté le 30 Juin 2010

classiques, Kodu peut exprimer les concepts avancés de conception de jeu d'une manière simple, directe et intuitive (voir figure 3.5).


FIGURE 3.5 – Kodu

Associé à cet outil, le « *Kodu Classroom Kit* » est disponible aux enseignants et fournit un ensemble de leçons et d'activités. Les leçons sont conçues pour être flexibles de manière à ce que l'enseignant puisse y trouver ce qu'il juge adapté à son contexte d'enseignement. À travers ces leçons, Kodu introduit la logique, la résolution de problèmes, les conditions et les séquences d'instructions avec une approche orientée objet. Il permet également de montrer que la programmation est un outil de création qui permet aux étudiants de travailler leur imagination. Ce Kit fournit également des enquêtes destinées aux enseignants et étudiants qui utilisent le jeu. Ces enquêtes à retourner aux concepteurs du jeu ont pour objectif de fournir un cadre d'analyse sur les expériences réalisées.

**Cleogo** [CB98] est un environnement de travail collaboratif qui permet à plusieurs personnes de développer simultanément des programmes à l'aide de trois métaphores de programmation : un langage de manipulation directe, un langage iconique et un langage à base de texte standard.

La programmation textuelle utilise le dialecte du langage Logo à l'exception des commandes de traitement des listes. Les lignes du programme sont exécutées au fur et à mesure de leur saisie. L'utilisateur peut ré-exécuter une ligne ou une séquence de lignes à l'aide d'un historique. La programmation iconique est réalisée à travers une interface représentant tous les éléments du langage utilisé par Cleogo (définition et appel de procédures, gestion des paramètres, utilisation de boucles et de conditions, calcul d'expressions). Toutes les actions réalisées à travers l'interface iconique sont retranscrites dans les modes de programmation à base de texte et de manipulation directe. La programmation à manipulation directe est plus limitée que les deux autres paradigmes. Elle permet seulement à l'utilisateur de contrôler la tortue à l'aide de la souris. Ce dernier peut donc déplacer la tortue et la faire avancer et reculer, tourner à gauche et à droite et (dés)activer le tracé. Chaque action est immédiatement répercutée dans les environnements iconiques et textuels. L'utilisateur peut également enregistrer des séquences d'actions pour créer des procédures mais ne peut les appeler ou traiter les conditions, les boucles et les expressions à travers l'environnement de manipulation directe. Toutefois, les enfants utilisateurs de ce paradigme ne semblent pas avoir de problème avec ce manque d'équivalence entre les différentes métaphores.

Le fait de proposer plusieurs paradigmes de programmation laisse la liberté à l'utilisateur de choisir celle qui lui correspond le mieux. Toutefois, quel que soit le paradigme choisi, chaque action d'un utilisateur est immédiatement communiquée à tous les participants. L'objectif de Cleogo est donc de permettre une collaboration en temps réel où chaque utilisateur est au courant des actions de ses collaborateurs et peut librement partager son système de contrôle. Chaque utilisateur a sa propre interface et peut prendre part à une collaboration via une connexion internet, auquel cas, une communication audio doit être établie pour permettre un échange naturel entre les participants. Les actions des différents utilisateurs sont réalisées en temps réel et peuvent conduire à des situations contradictoires. C'est alors aux participants de gérer le conflit par la communication. Le seul objectif du logiciel et d'assurer que chaque participant soit au courant des actions de autres personnes.

L'utilisation peut sembler désordonnée car chaque participant est libre de faire ce qu'il souhaite quand il le souhaite. Mais l'objectif de Cleogo porte justement sur cet aspect et pousse les utilisateurs à s'organiser pour résoudre un problème de manière cohérente.

**Compalgo** [42] (voir figure 3.6) est un environnement de programmation en langage algorithmique. Ce langage, typé et procédural, supporte la déclaration de structures, de fonctions et de procédures. Il permet aussi la programmation modulaire grâce à la définition et à l'importation de bibliothèques de programmes. Il intègre : un éditeur graphique multi-documents avec coloration syntaxique, une console d'exécution et de saisie et un débogueur. Compalgo est développé par des enseignements de l'IUT (Institut Universitaire de Technologie) 'A' Paul Sabatier Toulouse III. Il est utilisé dans une approche algorithmique à des fins d'initiation à la programmation pour les étudiants de première année de DUT Informatique.

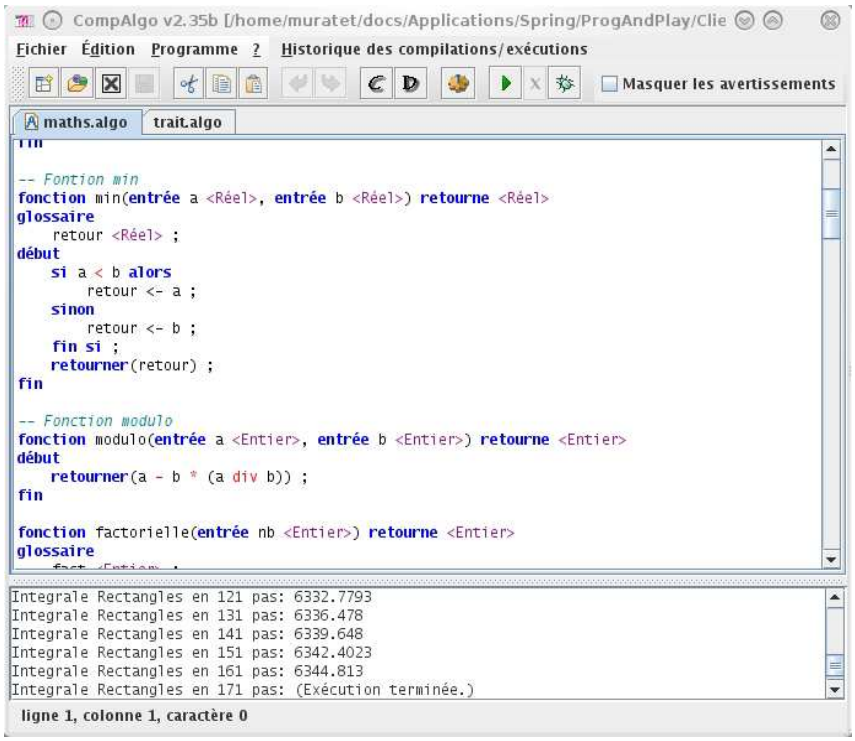



FIGURE 3.6 – Compalgo

Associé à cet outil, les bibliothèques et exemples nécessaires à la réalisation des travaux pratiques sont diffusées via la plateforme Moodle de l'IUT (Moodle est un environnement d'apprentissage libre basée sur une application Web gratuite que les acteurs de l'éducation peuvent utiliser pour créer des sites d'apprentissage autour de contenus et d'activités pédagogiques, ici Compalgo). Compalgo n'est donc pas à l'image des précédents exemples un langage de programmation à base de blocs mais fournit aux étudiants un environnement de programmation

42. `http://www.iut-tlse3.fr/moodle/course/view.php?id=1528` consulté le 30 Juin 2010

où le langage utilisé reflète la structure syntaxique et grammaticale du langage algorithmique utilisé en travaux dirigés.

**Conceptions d'exercices adaptés au centre d'intérêt des étudiants**

Stevenson et Wagner [SW06] ont analysé des manuels d'exercices et leurs usages historiques pour énoncer huit principes de base en vue de concevoir de « bons » exercices de programmation : être basé sur un problème du monde réel ; permettre aux étudiants de réaliser une solution réaliste à ce problème ; leur permettre de se concentrer sur un problème actuel du cours dans un contexte de programmes conséquents ; être stimulant ; être intéressant ; utiliser une ou plusieurs interfaces de programmation applicatives ; proposer plusieurs niveaux d'exercices qui permettent une optimisation des programmes ; permettre la créativité et l'innovation. Pour mettre en place ces exercices, les auteurs proposent une pédagogie par projet basée sur un robot d'indexation et un évaluateur de courriels indésirables. Greitzer *et al.* [GKH07], quant à eux, expliquent en particulier qu'une approche efficace consiste à encourager l'apprenant à travailler immédiatement sur des tâches réalistes et significatives.

Dans cette optique, une approche prometteuse consiste à utiliser la culture vidéo-ludique des étudiants pour les motiver, à travers un jeu vidéo, à investir du temps dans la pratique de la programmation. Deux orientations ont été explorées : développer un nouveau jeu vidéo ou jouer à un jeu vidéo.

Pour la première orientation, nous pouvons citer trois exemples. Chen et Cheng [CC07] proposent de demander aux étudiants, à travers un projet collaboratif, d'implémenter en C++ un mini jeu vidéo interactif en un semestre, à l'aide d'un cadre méthodologique. Gestwicki et Sun [GS08] ont réalisé une étude de cas sur le jeu EEClone. Ce jeu de type arcade est implémenté en Java : les étudiants analysent différents patrons de conception avec EEClone et à partir de cette expérience, étudient comment ils peuvent les intégrer dans leur propre jeu. Leutenegger et Edgington [LE07] utilisent directement les jeux vidéo pour enseigner les bases de l'informatique. Ces auteurs soutiennent que la programmation de jeux vidéo motive surtout les novices. Ils utilisent le développement de jeux 2D comme thème central.

La seconde orientation consiste à permettre à l'étudiant d'apprendre la programmation dans le contexte d'un jeu vidéo. La majorité des jeux existants de ce type permettent au joueur de programmer un robot afin de le faire combattre dans une arène contre un ou plusieurs robots programmés par d'autres joueurs. La bataille de robots est exécutée en temps réel sur l'écran

de l'ordinateur et la motivation de jeu émerge de la compétition entre les joueurs. Ces jeux sont conçus pour aider les étudiants à pratiquer un langage de programmation spécifique. Ces jeux sont accessibles à tout type de programmeur, des débutants (un simple robot peut être écrit en quelques minutes) aux experts (réaliser une véritable IA peut prendre plusieurs mois). A titre d'exemple, « *Robocode* » [43] [Har04] (voir figure 3.7) est compatible avec Java, « *Marvin's Arena* » [44] est compatible avec tous les langages .NET, « *Gun-Tactyx* » [45] utilise le langage SMALL et « *Robot Battle* » [46] utilise un langage de script spécifique. Dans ce type de jeu, l'activité de programmation est distincte de la simulation. Tout d'abord, le joueur écrit une IA pour son robot, puis l'exécute dans le contexte du jeu. Ainsi, le joueur est inactif durant la simulation et devient spectateur du déroulement de son programme.

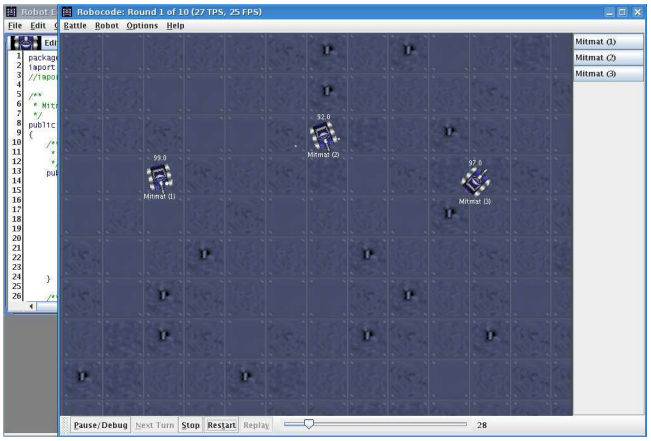



FIGURE 3.7 – Robocode : un simulateur de batailles de robots.

Toujours dans cette seconde orientation, *Colobot* [47] place le joueur dans le rôle d'un explorateur spatial dont l'objectif est d'explorer et coloniser des planètes à la recherche de matières premières nécessaires à sa survie. Le joueur peut progressivement construire et programmer de nouveaux robots pour l'aider dans ces différentes tâches et se protéger contre les créatures primitives et hostiles présentes sur certaines planètes. Ce jeu mise sur un scénario original pour motiver le joueur à résoudre les problèmes auxquels il se trouve confronté. Le langage de programmation utilisé est un langage orienté objet similaire au C++.

43. `http://robocode.sourceforge.net/` consulté le 8 Janvier 2010
44. `http://www.marvinsarena.com` consulté le 8 Janvier 2010
45. `http://apocalyx.sourceforge.net/guntactyx`consulté le 8 Janvier 2010
46. `http://www.robotbattle.com` consulté le 8 Janvier 2010
47. `http://www.ceebot.com/colobot/index-f.php` consulté le 8 Janvier 2010

C'est le seul exemple de jeu vidéo que nous connaissons qui intègre la programmation dans son scénario de manière interactive et cohérente avec l'univers du jeu (voir figure 3.8). Il inclut également une présentation pédagogique des concepts de programmation via une documentation détaillée consultable directement dans le contexte du jeu. *Colobot* est défini par ses concepteurs comme un jeu de STR. Pourtant, les jeux temps réel sont dynamiques et requièrent une forte interaction de la part du joueur alors que l'activité de programmation nécessite du temps de conception et de réalisation. Par conséquent, les règles classiques des jeux de STR ont été modifiées dans *Colobot* pour permettre cette intégration. Le joueur ne peut contrôler directement qu'une seule entité à la fois, d'où l'utilité d'automatiser ces robots à l'aide de programmes. Le mode multijoueur est absent pour éviter les temps d'attente entre joueurs pendant les phases de programmation.


FIGURE 3.8 – Colobot : un jeu sérieux pour l'apprentissage de la programmation.

### Autres approches

La *RoboCup* [48] (voir figure 3.9) est une initiative éducative de recherche internationale. Son but est de regrouper la recherche en intelligence artificielle et en robotique. Cette manifestation intègre et examine une gamme étendue de technologies à travers trois compétitions : la *RoboCupSoccer*, la *RoboCupRescue* et la *RoboCupJunior*. Cette initiative diffère des batailles de robots en raison de son rapprochement avec la robotique (programmation de robots physiques) et de par les objectifs de jeux qui ne portent pas sur la destruction de robots adverses. La

48. `http://www.robocup.org/Intro.htm`

*RoboCupSoccer* consiste à confronter des robots réels ou simulés dans des matchs de foot. Différentes ligues et modes de jeu ont été imaginé, en vue de permettre la participation d'un maximum de disciplines scientifiques, pour traiter de problèmes sur les environnements dynamiques et la manipulation d'informations incomplètes ou bruitées. La *RoboCupRescue* est une nouvelle compétition mise en place pour apporter plus de diversité. Elle est complémentaire de la *RoboCupSoccer* en proposant des défis dans la logistique et la planification à long terme pour la collaboration d'agents hétérogènes et d'équipes d'agents. Enfin, pour les jeunes, la *RoboCupJunior* est une compétition fortement orientée sur l'éducation et destinée à fournir une introduction excitante aux caractéristiques de la robotique.

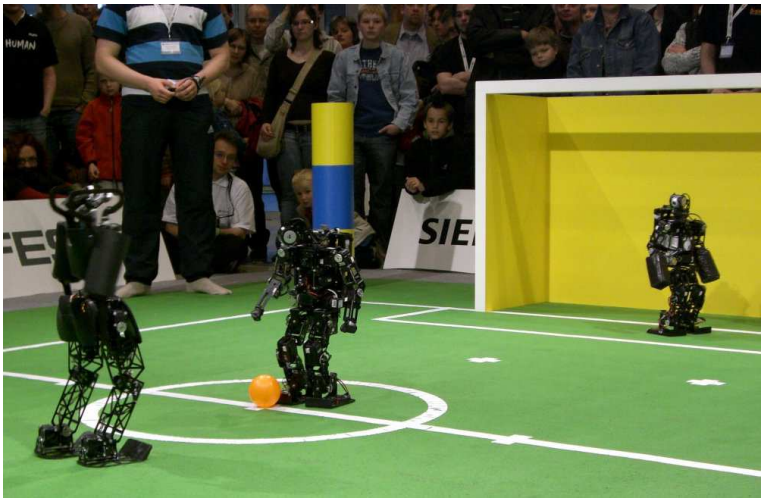


FIGURE 3.9 – La *RoboCup* 2007 (Allemagne).

WISE (*Wireless Intelligent agent Simulation Environment*) [CHHY04] présente également une approche originale difficile à classer. Cet outil se trouve à cheval entre le jeu vidéo et la manipulation de robots physiques à l'image de la *RoboCup*. WISE combine une version virtuelle et physique du jeu *Wumpus World*. Il permet une distribution physique d'agents (humain ou robot) pour un jeu interactif et fournit un environnement dynamique d'apprentissage qui permet la mise en application d'un grand nombre de concepts informatiques. Le jeu peut être utilisé comme un outil de démonstration lors de cours magistraux et peut permettre aux étudiants, lors de travaux dirigés, de travailler sur des exercices pour tester, étendre, ou modifier le simulateur. WISE peut être joué de manière coopérative ou compétitive et nécessite, pour la version physique, un grand nombre de ressources (espace, robots, etc.). Ce jeu a été utilisé pour l'enseignement de l'intelligence artificielle, du multimédia et des réseaux.

Un dernier exemple sortant de l'ordinaire est le jeu de société « c-jump » (voir figure 3.10). Ce jeu de plateau permet de découvrir les fondamentaux de la programmation comme la séquence, les structures de contrôle conditionnelles et itératives et le concept de variables. Chaque joueur contrôle un pion (un nivoplanchiste ou *snowboardeur*) et doit être le premier à atteindre l'arrivée (*i.e.* avoir parcouru tout le programme).

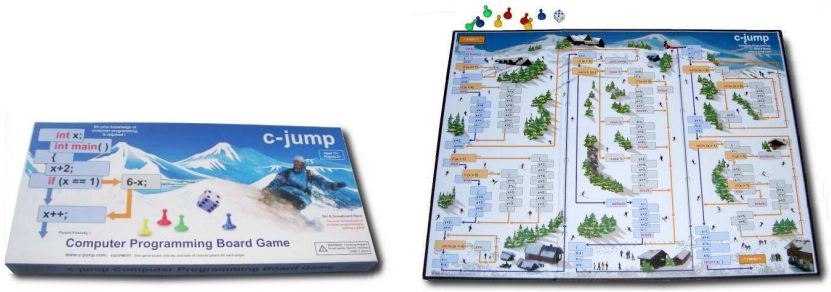



FIGURE 3.10 – C-jump : un jeu de plateau pour découvrir les fondamentaux de la programmation.

# Conclusion

Malgré l'omniprésence de l'informatique dans nos sociétés, les nouveaux étudiants sont peu attirés par cette discipline. La raison de ce phénomène semble résider dans la difficulté qu'ont les formations à proposer des contenus en lien direct avec les applications du monde courant. De ce fait, les étudiants peinent à se projeter dans l'avenir à travers une telle présentation de la formation.

Pour résoudre ce problème, le rapport de l'ACM/IEEE [AIC08] énonce un certain nombre de recommandations :

- ne pas changer la nature fondamentale de la discipline, c'est-à-dire ne pas simplifier la formation ou réduire son niveau académique ;
- proposer une formation qui soit intéressante et attractive aux étudiants actuels et qui leur fournisse les connaissances et compétences requises pour une carrière en informatique ;
- reconnaître que la « *crise de l'informatique* » est un problème important pour la communauté informatique. De nombreuses recherches et expérimentations actuelles portent sur ce problème, mais il est important d'encourager et de soutenir de nouvelles tentatives à ce sujet ;

– ne pas proposer de conseils reconnus comme étant efficaces dans des environnements spécifiques mais faire des propositions génériques. De cette manière, différents groupes de la communauté seront encouragés à chercher des solutions appropriées à leurs étudiants.

Parmi les propositions présentées dans ce chapitre, l'approche basée sur les jeux vidéo est retenue, car elle semble avoir un potentiel d'attraction et de motivation important auprès des générations actuelles.

# Conclusion de l'état de l'art

Le jeu sérieux permet donc une immersion et une interaction avec un monde virtuel qui peut servir de support à une formation. La composante ludique du jeu permet de motiver le joueur et le maintient dans une dynamique d'apprentissage. Le jeu sérieux peut donc prendre une place importante et s'imposer comme un complément aux méthodes de formation classiques. De plus son utilisabilité dans la plupart des secteurs d'activités lui donne un avantage certain pour son devenir. Les principaux travaux sur les jeux sérieux, pour permettre leur essor et leur développement, doivent porter sur l'infrastructure, la conception de jeux cognitifs et l'immersion (Michael Zyda [Zyd05]).

- l'infrastructure consiste à développer les MMO, les moteurs de jeux et leurs outils, le *streaming* (*streaming* : technique consistant à transmettre des données aux utilisateurs et à les rendre disponibles au fur et à mesure de leur arrivée pour ainsi éviter l'attente d'un téléchargement complet), les consoles de jeux nouvelles générations, le « sans fil » et les technologies mobiles ;
- la conception de jeux cognitifs doit s'appuyer sur la modélisation et la simulation des émotions et des comportements humains. Elle doit également analyser et innover sur des styles et genres de jeux nouveaux. Enfin, l'intégration d'une pédagogie dans l'histoire des jeux sérieux reste un point important à approfondir ;
- l'immersion doit être améliorée à travers le graphisme, le son, l'haptique, mais aussi en utilisant les émotions et l'état du joueur.

Dans ce sens, les jeux sérieux peuvent apporter leur contribution à la résolution de la « *crise de l'informatique* » dans l'enseignement. Cette orientation a d'ailleurs été présentée à travers les batailles de robots et *Colobot* qui peuvent être considérés comme des jeux sérieux pour l'apprentissage de la programmation (voir définition section 1.2).

Parallèlement à ces outils clairement définis, les moteurs de jeux de STR à code source ouvert présentent des caractéristiques intéressantes pour aborder le problème de la motivation

des étudiants en informatique. En effet leur interface de programmation d'IA combinée à la possibilité de modifier le contenu du jeu doit permettre la conception de jeux de STR où la pratique de la programmation aurait son importance. Il serait alors possible de proposer aux étudiants des jeux sérieux pour l'apprentissage de la programmation plus proches des standards des jeux vidéo.

# Deuxième partie

# Contributions

# 4

# Conception d'un jeu sérieux de STR pour l'apprentissage des fondamentaux de la programmation

Dans le cadre de ce travail de recherche, l'objectif principal consiste à concevoir, réaliser et évaluer un jeu sérieux de STR pour l'apprentissage de la programmation. Le choix de ce type de jeu comme support au jeu sérieux a été présenté dans le chapitre 2 et se justifie par les trois composantes suivantes : un environnement complexe propice à la mise en place de problèmes informatiques, une popularité auprès des joueurs et une disponibilité de moteurs de jeu à code source ouvert réutilisables.

La conception du jeu sérieux a suivi trois étapes principales. La première a consisté à vérifier la popularité des jeux de STR auprès d'un échantillon d'étudiants apprenant la programmation et de définir le principe général du jeu sérieux. La deuxième phase a regroupé l'ensemble des réalisations techniques qui permettent l'intégration de la programmation dans les jeux de STR. Enfin la dernière étape a consisté à concevoir le jeu à proprement dit en y intégrant de manière cohérente le savoir à enseigner.

## 4.1 Cadrage du jeu sérieux

Cette section s'attache donc à vérifier la popularité des jeux de STR auprès des étudiants susceptibles d'utiliser le jeu sérieux. En effet, pour espérer bénéficier du potentiel motivationnel

du jeu vidéo, il est préférable de baser le jeu sérieux sur un genre apprécié des étudiants. Enfin, le fonctionnement général du jeu est présenté avec notamment l'intégration de la programmation dans l'activité de jeu.

### 4.1.1 Popularité des jeux de STR auprès des étudiants

La popularité des jeux de STR a été vérifiée par trois séries d'enquêtes réalisées de septembre 2007 à juin 2010 auprès de 900 étudiants répartis dans huit formations différentes, soit 59% des inscriptions pédagogiques. Les formations choisies pour cette enquête proposent toutes un enseignement des fondamentaux de la programmation. La pratique du jeu vidéo pour ces étudiants en fonction du sexe et du type de formation est synthétisé dans la figure 4.1. Il apparaît sur ces graphiques, que les étudiants des formations IUT sont plus « joueurs » que les étudiants de la faculté. De même, le public féminin est moins sensibilisé aux jeux vidéo que le public masculin. Il importe toutefois de préciser que, dans ces enquêtes, les étudiantes représentent seulement 19% de la population. Avec 77% de joueurs parmi l'ensemble des étudiants interrogés, ces données corroborent les chiffres de l'ESA [Ass09] et illustrent la popularité des jeux vidéo auprès des étudiants d'informatique.

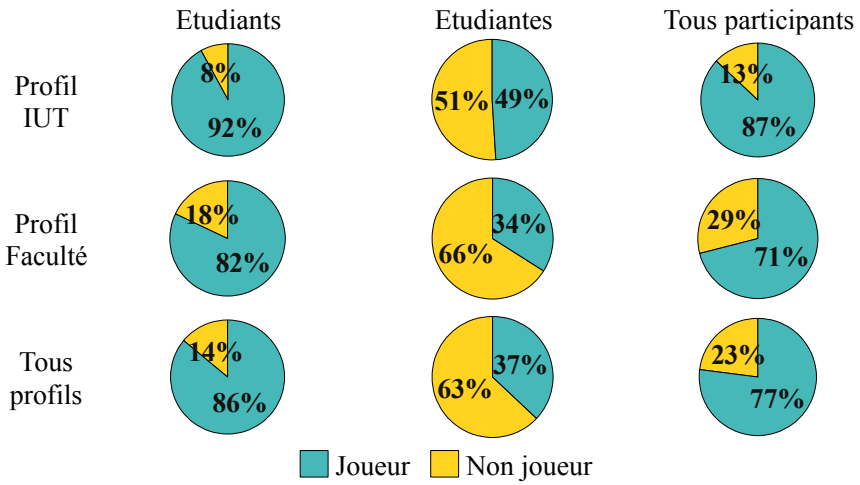



FIGURE 4.1 – Pourcentage de joueurs en fonction du sexe et du type de formation.
*Enquête réalisée auprès de 900 étudiants dont 730 garçons et 170 filles sur les 1533 inscriptions pédagogiques des huit formations interrogées.*

La première série d'enquêtes concerne 181 étudiants (154 garçons et 27 filles) dans trois formations différentes (deux en informatique et une en génie civil). Dans ce questionnaire, neuf

familles de jeu sont proposées et chaque étudiant joueur indique les types de jeu pratiqués. La figure 4.2 met en évidence le pourcentage de joueurs pour chaque type de jeu. Notons que le jeu de STR arrive en tête et se trouve être également joué par les filles (57% des filles joueuses jouent aux jeux de stratégie).

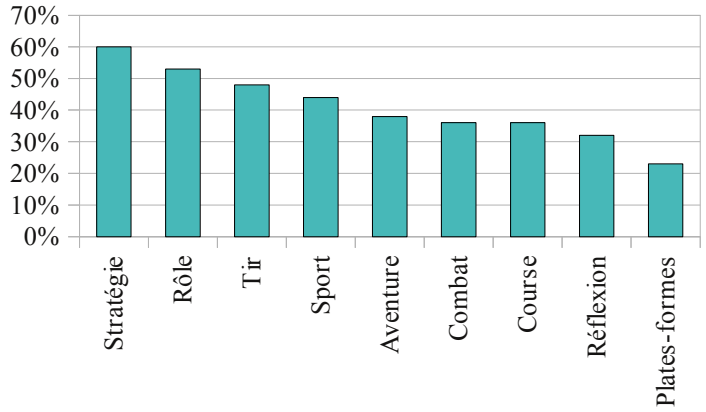



FIGURE 4.2 – Enquête 1 : Pourcentage de joueurs en fonction du type de jeu.
*Enquête réalisée auprès de 181 étudiants dont 154 garçons et 27 filles.*

La deuxième série d'enquêtes concerne 113 étudiants d'informatique (78 garçons et 35 filles). L'objectif est d'affiner la perception de la pratique du jeu vidéo chez les étudiants. Pour cela, les étudiants joueurs attribuent une note de 1 à 7 aux neuf familles de jeux proposées. La note maximale indique qu'un type de jeu est particulièrement apprécié d'un étudiant alors que la note minimale dénote le contraire. Il est donc possible de déterminer trois classes d'appréciation en fonction de la répartition des notes des étudiants : les types de jeu « non populaires » sont ceux dont le nombre maximum d'étudiants se situe sur les notes 1 et 2 ; les types de jeu « indifférents » sont ceux dont le nombre maximum d'étudiants se situe sur les notes 3, 4 et 5 ; les types de jeu « populaires » sont ceux dont le nombre maximum d'étudiants se situe sur les notes 6 et 7. À la vue des données récoltées aucun type de jeu n'entre dans la première classe des jeux « non populaires ». La figure 4.3 présente les types de jeu entrant dans la classe « indifférents » et la figure 4.4 présente les types de jeux de la classe « populaire » (les jeux de STR font partie de ce groupe). Il est apparu que la classification proposée était parfois mal interprétée par les étudiants qui éprouvaient donc des difficultés à évaluer leur intérêt pour chaque famille de jeu. Par conséquent, une troisième série d'enquêtes a donc été réalisée.

Dans cette dernière version du questionnaire, les étudiants joueurs doivent simplement lister leurs cinq jeux préférés. Chaque jeu est ensuite classé suivant la nomenclature du magazine

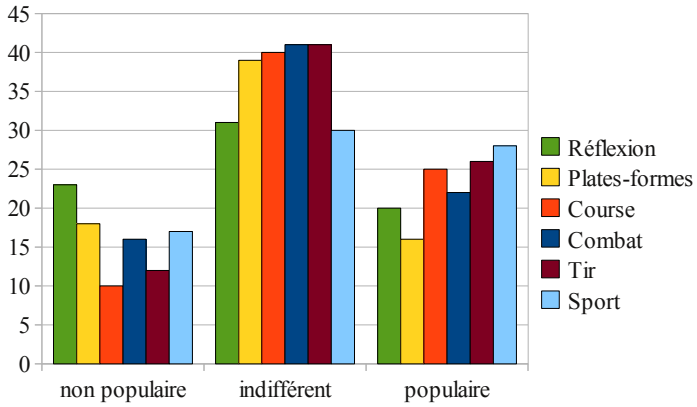



FIGURE 4.3 – Enquête 2 : Répartition de l'intérêt des étudiants pour les jeux « indifférents ».
*Enquête réalisée auprès de 113 étudiants dont 78 garçons et 35 filles.*

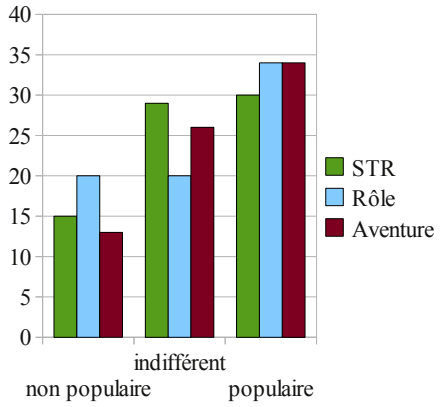



FIGURE 4.4 – Enquête 2 : Répartition de l'intérêt des étudiants pour les jeux « populaires ».
*Enquête réalisée auprès de 113 étudiants dont 78 garçons et 35 filles.*

spécialisé *GameSpot* présenté dans la section 1.1.2. La classification de *GameSpot* a été choisie en raison de sa visibilité (magazine de jeux vidéo le plus consulté sur Internet) et de la présence sur le site Web d'une base de données importante accompagnée d'un moteur de recherche qui facilite le classement de chaque jeu vidéo. Lors de ces enquêtes, 606 étudiants (498 garçons et 108 filles) de cinq formations en informatique ont été interrogés. 321 jeux différents ont été identifiés parmi les 1891 occurrences de jeux cités. Afin d'augmenter la lisibilité des informations récoltées, seules les catégories mentionnées par plus de 10% des étudiants joueurs sont présentées dans les résultats. Il apparaît que les jeux de STR sont mentionnés par 36% des étudiants (voir figure 4.5). Ils occupent donc une place importante et se positionnent comme le troisième type de jeu préféré parmi les 36 catégories de la classification de *GameSpot*. Pour les filles joueuses, les jeux de STR sont mentionnés pour 17% d'entre elles ce qui place ce type de jeu en sixième position (voir figure 4.6).

La première série d'enquêtes a permis d'obtenir une vision exhaustive de la pratique du jeu vidéo par les étudiants en fonction d'une classification donnée. Dans cette approche, les étudiants doivent déterminer les catégories correspondantes à leurs jeux préférés. La deuxième série d'enquête a permis d'obtenir une vision plus précise et a montré qu'aucun type de jeu ne fait réellement l'unanimité entre les étudiants. En effet les trois classes d'appréciation sont toujours représentées de manière non négligeable pour chaque type de jeu. Le point critique de la

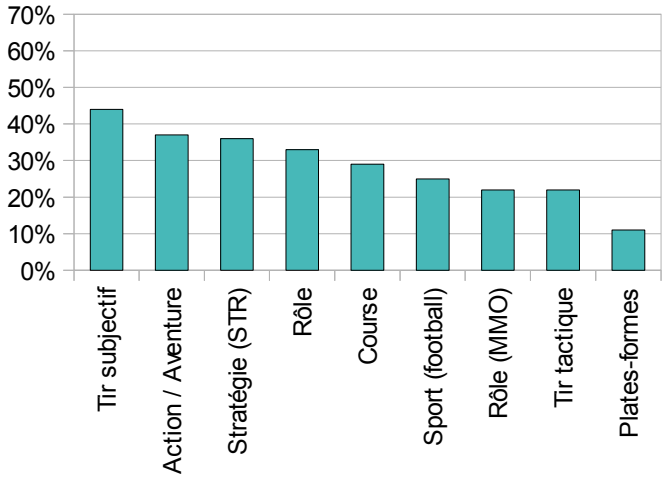



FIGURE 4.5 – Enquête 3 : Les neuf catégories de jeu les plus mentionnés parmi la classification de *GameSpot*.

*Enquête réalisée auprès de 606 étudiants dont 498 garçons et 108 filles.*

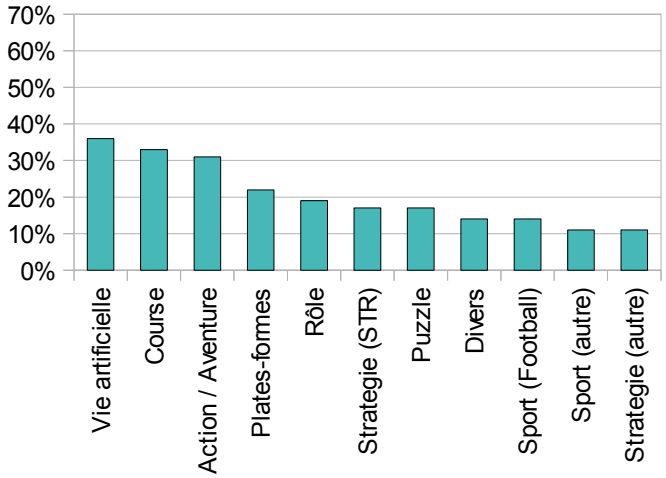



FIGURE 4.6 – Enquête 3 : Les onze catégories de jeu les plus mentionnés par les étudiantes joueuses parmi la classification de *GameSpot*.

*Extraction des données pour les 108 étudiantes de la troisième série d'enquêtes.*

méthode utilisée lors de ces deux premières enquêtes porte sur la terminologie employée pour caractériser les différents types de jeux et sur la difficulté de certains étudiants à projeter leurs jeux dans la classification proposée. La troisième série d'enquêtes avait donc pour objectif de remédier a ce problème en déchargeant les étudiants de cette tâche et en transférant l'activité

de classement lors de l'analyse des données. Mais cette approche pose le problème de la complétude des informations et de la capacité à déterminer de manière exhaustive les types de jeux appréciés des étudiants à partir d'un échantillon de jeux.

L'analyse croisée de ces différentes enquêtes a tout de même permis de vérifier la bonne popularité des jeux de STR auprès des étudiants en informatique. Par conséquent, le choix initial d'un jeu de STR comme support au jeu sérieux s'est avéré être en lien direct avec les centres d'intérêts des étudiants susceptibles d'utiliser le jeu. Mais, en quoi consiste réellement ce jeu sérieux ? La section suivante aborde cette question à travers une présentation générale du jeu sérieux.

### 4.1.2 Fonctionnement général du jeu sérieux

Afin d'évaluer l'intérêt du jeu sérieux dans différentes situations d'enseignement, une étape incontournable est de rechercher et convaincre des enseignants d'accepter de déployer le jeu dans leur formation. L'utilisation d'un jeu sérieux adaptable à différents contextes, c'est-à-dire à différents langages et paradigmes de programmation, différentes méthodes pédagogiques et différents étudiants doit faciliter l'adhésion des enseignants. L'analyse des outils existants dédiés à l'apprentissage de la programmation a permis d'identifier les batailles de robots et *Colobot* comme étant des jeux sérieux proches de jeux de STR (voir section 3.2.2). Malgré leurs qualités propres, ces jeux ne sont pas utilisables dans différents contextes d'enseignements puisqu'ils reposent sur un langage de programmation spécifique qui peut ne pas correspondre aux choix pédagogiques des enseignants. Il devient alors pertinent de concevoir un jeu sérieux gratuit et portable. La gratuité est une caractéristique supplémentaire pour favoriser l'utilisation du logiciel dans un milieu universitaire et éviter que des problèmes d'ordre financier soient un frein à la réalisation des expérimentations.

Les jeux de STR sont des programmes extrêmement complexes, composés de plusieurs dizaines de milliers de lignes de code. L'objectif n'étant pas de réaliser un nouveau jeu de STR, l'utilisation d'un moteur de jeu existant s'est imposé. Ce moteur doit avoir son code source ouvert pour permettre d'y intégrer les fonctionnalités liées au jeu sérieux.

Dans les jeux de STR, le joueur donne des ordres à ses unités pour réaliser des actions comme se déplacer, construire, ou attaquer. Ces ordres sont donnés en cliquant sur la carte à l'aide de la souris. Dans la mesure où ces ordres peuvent être remplacés par des instructions

de programmation, puis enchaînés et ordonnés pour être transmis au moteur de jeu, la transformation d'un STR en jeu sérieux pour l'apprentissage de la programmation devient possible.

Suite à l'analyse des systèmes existants, tels que *Colobot* et les batailles de robots, une méthode d'interaction compatible avec les jeux de STR et la pratique de la programmation est proposée. Cette méthode s'articule autour de deux phases. Dans un premier temps le joueur/étudiant conçoit des programmes capables de contrôler ses unités en vue de leur attribuer des comportements. Cette phase de conception est réalisée à la manière des jeux de bataille de robots où le joueur/étudiant écrit son IA puis la teste dans le contexte du jeu. Puis, dans un second temps, une fois ses programmes mis au point, le joueur/étudiant les utilise dans les différents modes de jeu (campagne et escarmouche). Tout en conservant une interaction directe sur l'environnement, le joueur/étudiant peut exécuter ses programmes en cours de partie à la manière de *Colobot*. Ce modèle peut être facilement porté à N étudiants en exploitant le composant multijoueur des jeux de STR où chaque participant est libre de concevoir des programmes et de les utiliser en cours de partie s'il les juge pertinents (voir figure 4.7).

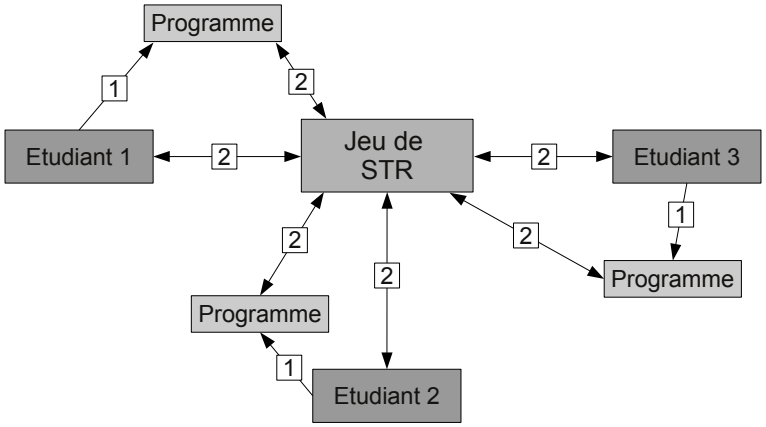



FIGURE 4.7 – Interaction entre chaque joueur, leur programme et le jeu de STR.
*Phase 1 : Codage*
*Phase 2 : Interaction*

L'activité de jeu consiste alors à écrire des programmes plus ou moins sophistiqués destinés à commander des unités, ouvrant ainsi la voie à différents niveaux de complexité. L'utilisation de la programmation peut devenir un ressort qui renforce l'apprentissage, car elle apporte un intérêt nouveau au jeu en permettant au joueur de dépasser les contraintes d'interaction liées aux périphériques d'entrée/sortie et à ses compétences physiques. La programmation peut lui permettre de déléguer certaines tâches à ses programmes afin de se concentrer sur les aspects

du jeu les plus intéressants. Les spécifications de notre jeu sérieux sont donc les suivantes : véhiculer des concepts de programmation, être portable dans différents contextes d'enseignement, être basé sur un jeu de STR, conserver le *gameplay* du jeu de STR support et autoriser le contrôle des unités via des programmes informatiques.

Pour permettre la communication entre le programme du joueur et un jeu de STR, le système Prog&Play a été développé.

## 4.2 Spécification du système Prog&Play

L'objectif du système Prog&Play est de permettre au joueur de réaliser des programmes capables d'interagir avec un jeu de STR. De cette manière, le joueur pourra suivre le déroulement de son programme à travers le comportement de ses unités dans le jeu vidéo. Chaque moteur de jeu, présenté dans la section 2.4, propose un langage de script et une interface associée pour permettre aux joueurs de personnaliser le jeu ou de créer leur propre IA. Or, ces langages de script peuvent ne pas correspondre aux approches choisies par les enseignants pour aborder les fondamentaux de la programmation. Pour cette raison, le système Prog&Play est conçu comme une interface entre les moteurs de STR et les langages de programmation. Parmi un ensemble de langages et un ensemble de moteurs compatibles Prog&Play, chaque enseignant est libre de choisir les deux composantes les plus adaptées à son approche pédagogique (voir figure 4.8).

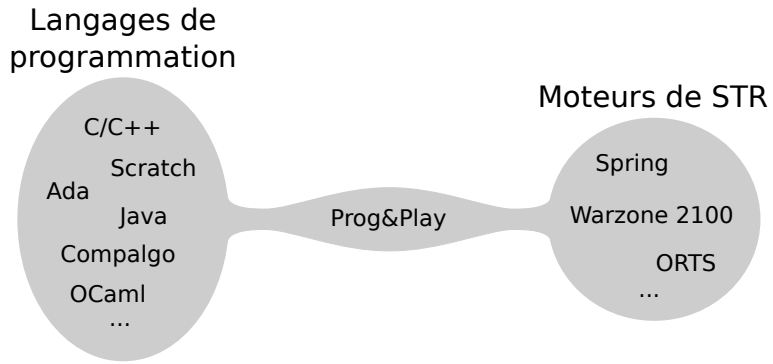



FIGURE 4.8 – Prog&Play : une interface entre les langages de programmation et les moteurs de STR.

Prog&Play est donc disponible sous la forme d'une API (*Application Programming Interface*) utilisable par les joueurs pour concevoir des programmes compatibles avec les jeux de STR qui intègrent le système.

### 4.2.1 Première version

L'API Prog&Play, telle qu'elle a été utilisée pour les expériences, est l'évolution d'un premier système basé sur l'utilisation d'une bibliothèque dynamique et de la conception d'un environnement de développement dédié. La section suivante présente ce système et pointe les limites qui ont conduit à concevoir l'API Prog&Play.

#### Vue d'ensemble

Cette première version, structurée autour d'une bibliothèque dynamique, est composée de trois entités : le moteur de jeu, la bibliothèque dynamique contenant le code du joueur et le centre de développement.

Dans ce système (dont l'architecture fonctionnelle des différents composants est présentée dans la figure 4.9), le moteur du jeu doit être modifié pour permettre le chargement de la bibliothèque (à travers le « Chargeur ») et fournir aux joueurs les outils nécessaires à la manipulation des données (grâce à l'« IMJ » - « Interface du Moteur de Jeu »). La bibliothèque dynamique possède une interface (« IGCU » - « Interface de Gestion du Code de l'Utilisateur ») qui permet son utilisation et un *thread* (ou processus léger) pour l'exécution du code du joueur.

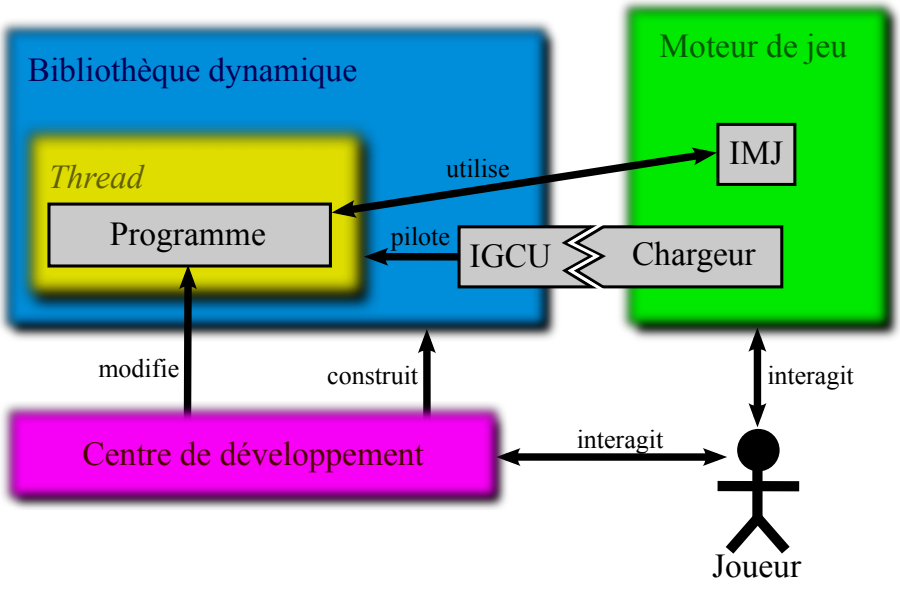



FIGURE 4.9 – Architecture fonctionnelle (version 1).

L'« IMJ » assure une double fonctionnalité. Premièrement, c'est une interface fournissant à l'utilisateur la possibilité d'interagir avec le jeu, elle sert de point d'entrée vers le moteur. L'ensemble des fonctions accessibles par cette interface doit être en relation avec les connaissances du joueur, les objectifs pédagogiques, etc. Deuxièmement, elle assure la synchronisation entre le code du joueur et le moteur du jeu, ceci afin de respecter l'intégrité et la consistance du déroulement de la partie.

L'« IGCU » permet de contrôler l'exécution du code de l'utilisateur à travers deux points d'entrée, les fonctions :

`void IGCU_Start(IMJ* data)` et `void IGCU_Stop(void)`.

Ces deux fonctions permettent respectivement de lancer l'exécution de l'IA dans un *thread* en lui indiquant les données utilisables à travers une « IMJ » et de stopper proprement l'exécution de ce *thread* contenant l'IA.

Le « Chargeur » est conçu pour charger la bibliothèque dynamique lorsque celle-ci est créée ou modifiée. Il a également la responsabilité de la libérer de la mémoire si celle-ci est supprimée du système de fichier. Le « Chargeur » a une autre utilité : il pilote l'exécution du code de l'utilisateur à travers les points d'entrée de l'« IGCU » afin de maintenir la même version entre la bibliothèque et le code exécuté. L'algorithme 1 présente le fonctionnement du « Chargeur ».

Finalement, le centre de développement permet au joueur de modifier son code contenu dans la bibliothèque et de (re)construire celle-ci pour indiquer au moteur que le code a changé et qu'il est temps de le mettre à jour.

**Bibliothèque dynamique**

La sémantique du terme bibliothèque dynamique résume bien son utilité. La bibliothèque fournit des fonctions qui peuvent être appelées et exécutées par le programme qui la consulte. Quant à la notion de dynamique, elle indique que la bibliothèque pourra être chargée, utilisée et éliminée pendant l'exécution du programme.

Dans cette première version, la bibliothèque contient le code saisi par le joueur et définit le comportement de ses unités. Le jeu utilise donc cette bibliothèque pour déterminer les actions à réaliser. À chaque modification de la bibliothèque, la nouvelle IA est rechargée. Grâce à ce principe, le code contenant le comportement des unités est complètement indépendant du jeu. Le joueur peut donc maintenant modifier son code et le recompiler sous forme d'une bibliothèque pour qu'il soit automatiquement intégré au jeu en cours de partie. En règle générale, la

**Algorithme 1** Gestion de la bibliothèque dynamique

```
tantque le jeu est en fonctionnement faire
  si la bibliothèque dynamique existe alors
    si la bibliothèque dynamique a été mise à jour alors
      si IGCU_Stop est chargé alors
        Stopper l'IA en cours d'exécution (IGCU_Stop)
      finsi
      Charger les points d'entrée (IGCU_Start et IGCU_Stop)
      si les points d'entrée ont été correctement chargés alors
        Lancer l'exécution de l'IA (IGCU_Start)
      sinon
        Libérer les points d'entrée (IGCU_Start et IGCU_Stop)
      finsi
    finsi
  sinon
    si IGCU_Stop est chargé alors
      Stopper l'IA en cours d'exécution (IGCU_Stop)
    finsi
  finsi
fin tantque
si IGCU_Stop est chargé alors
  Stopper l'IA en cours d'exécution (IGCU_Stop)
finsi
```

bibliothèque se suffit à elle même. Pourtant, dans notre application, elle est censée accéder à la boite à outils du moteur afin de manipuler les données.

L'exécution de la bibliothèque est réalisée dans un *thread* pour permettre au client de rester actif et apte à réagir aux actions de l'utilisateur ou du serveur. Dorénavant, des IA complexes peuvent être mises en place sans influencer les performances du jeu. En contre partie, la programmation parallèle introduit des difficultés supplémentaires pour fiabiliser le système. En effet, ce type de programmation demande la mise en place de mécanismes entre les différents processus pour permettre d'assurer la synchronisation et la cohérence des données partagées tout en étant transparent pour le joueur.

La fiabilité du système est également assurée en le protégeant contre les bogues générés par les étudiants. En effet, les joueurs étant en apprentissage de la programmation, il est fortement probable que ceux-ci réalisent des erreurs. Ces bogues peuvent causer des erreurs systèmes (erreur de segmentation par exemple) ou des levées d'exceptions. Pour récupérer les erreurs dé-

clenchées dans la bibliothèque les signaux systèmes sont utilisés. Ainsi, lorsqu'une interruption est générée dans le code du joueur, seul le *thread* exécutant le code du joueur est interrompu. Une information est alors donnée au joueur pour l'informer du type d'erreur ayant arrêté son IA. De cette manière, le fonctionnement du jeu n'est pas dépendant des mauvais fonctionnements de l'IA.

L'utilisation de la bibliothèque dynamique possède un avantage supplémentaire. Elle dissimule au joueur la complexité du jeu vidéo. En règle générale, vouloir modifier une partie d'un programme consiste, au préalable, à analyser la structure, l'organisation et le fonctionnement de l'application. La bibliothèque dynamique aide le joueur en extrayant l'IA du jeu. De cette manière, l'utilisateur n'a pas conscience des difficultés liées à l'intégration de son code dans le moteur de jeu.

### Environnement de développement

La bibliothèque dynamique donne au joueur l'opportunité de modifier son code de façon interactive. Elle l'assiste dans son travail en cachant la complexité du moteur. Cependant, l'utilisation du jeu reste fastidieuse en raison de l'arborescence et des dépendances des fichiers (emplacement des fichiers sources et de la bibliothèque dynamique générée, options de compilation, etc.). Dans ces conditions, il était nécessaire de rendre le logiciel plus intuitif en créant un environnement de développement appelée le centre de développement (CDD) (figure 4.10).

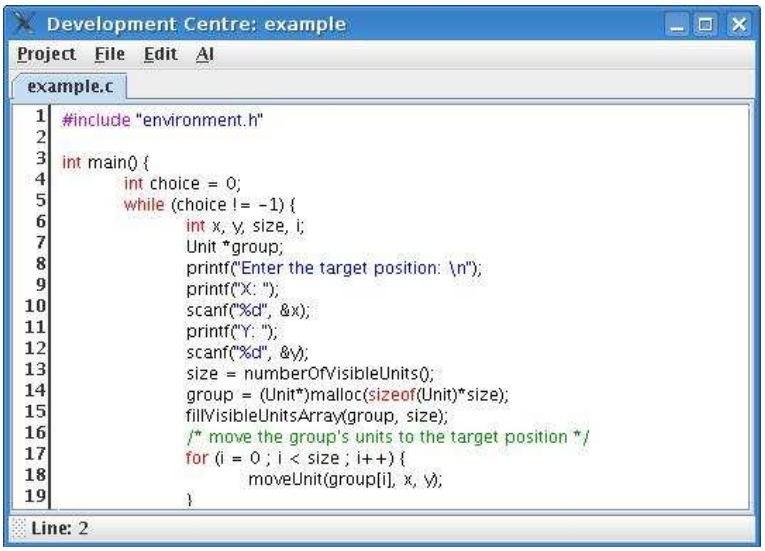



FIGURE 4.10 – Le centre de développement.

Le CDD est un composant qui facilite la conception et la manipulation des réalisations du joueur. Il est complémentaire du jeu et simplifie la gestion des projets à travers un ensemble de menus. Cependant, le CDD est indépendant et n'est pas nécessaire au fonctionnement du moteur et vice versa. Il complète la bibliothèque dynamique en simplifiant la manipulation du système de synchronisation et de liaison entre le code du joueur et le jeu. Ainsi, le joueur peut alors utiliser les fonctions de l'« IMJ » pour manipuler les entités du jeu et implémenter la classique fonction « `int main (){...}` » comme si son code était indépendant du jeu. Lors de la compilation, cette fonction est renommée en « GCU_exec » et utilisé par l'« IGCU » pour lancer l'exécution de l'IA. Grâce au CDD, le joueur peut facilement compiler et injecter son code dans le moteur afin d'observer les résultats.

### Critiques

Cette solution permet de stocker le programme compilé du joueur dans une bibliothèque chargée dynamiquement au cours de l'exécution du jeu. Moyennant quelques optimisations l'utilisation de bibliothèques dynamiques permet de satisfaire les contraintes suivantes :

1. permettre aux joueurs d'intégrer leur code au jeu dynamiquement ;
2. protéger le jeu contre les bogues venant des programmes des joueurs ;
3. cacher la complexité du moteur de jeu ;
4. être compatible avec différents langages de programmation.

Cependant, certains langages dits « interprétés » ne permettent pas la génération de bibliothèques dynamiques. Par conséquent, le point (4) n'est satisfait que pour un sous-ensemble des langages de programmation. Dans ce cadre, une seconde version nommée Prog&Play a été développée pour s'interfacer avec ce type de langages.

Concernant le CDD, l'aide à la conception et à la manipulation introduite par cet éditeur se trouve être insuffisante pour compenser les performances et fonctionnalités des environnements de développement dédiés à chaque langage de programmation. D'autre part, en contexte réel, les enseignants font un choix de langage et d'éditeur associé. L'idéal est donc de pouvoir utiliser l'API Prog&Play via les environnements de programmation choisis par les enseignants. Pour ces deux raisons, le principe d'un environnement de développement dédié au jeu a été abandonné.

### 4.2.2 Seconde version

Dans la nouvelle version, le programme de l'étudiant compilé ou interprété est indépendant du moteur de jeu et communique avec le jeu via une mémoire partagée.

#### Description générale

Les mémoires partagées sont utilisées comme méthodes de communication interprocessus c'est-à-dire comme un moyen d'échanger des données entre programmes fonctionnant en même temps. L'API Prog&Play sert d'interface à la gestion de cette mémoire partagée pour simplifier la communication et la synchronisation entre le programme du joueur et le moteur de jeu. Deux interfaces sont donc disponibles : la partie « Client » est utilisée par le joueur pour développer son propre programme et la partie « Fournisseur » est intégrée au moteur du jeu. Sur demande du programme du joueur, les données pertinentes du jeu sont copiées dans la mémoire partagée. Pour éviter des situations incohérentes, le programme du joueur travaille sur cette copie. Le programme du joueur lit les données et écrit les commandes à travers l'interface « Client ». La mémoire partagée est régulièrement vérifiée par le moteur de jeu pour réaliser les actions en attente. De cette manière, à n'importe quel instant, le joueur peut stopper l'exécution de son programme, le modifier et le relancer pour qu'il se reconnecte à la mémoire partagée sans perturber le déroulement du jeu. Cette architecture fonctionnelle est présentée dans la figure 4.11.

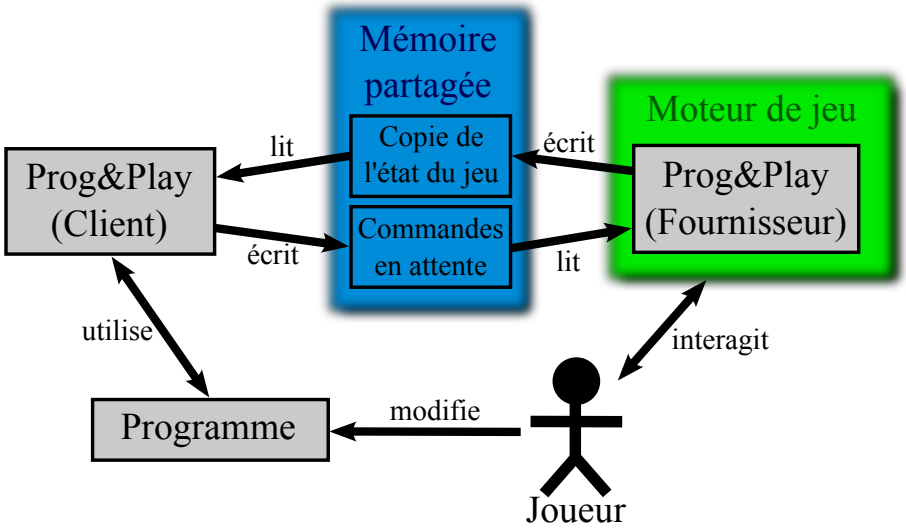



FIGURE 4.11 – Architecture fonctionnelle (version 2).

Le « Fournisseur » a donc en charge la gestion de la mémoire partagée. La figure 4.12 présente la séquence d'interaction entre le « Fournisseur » et la mémoire partagée.

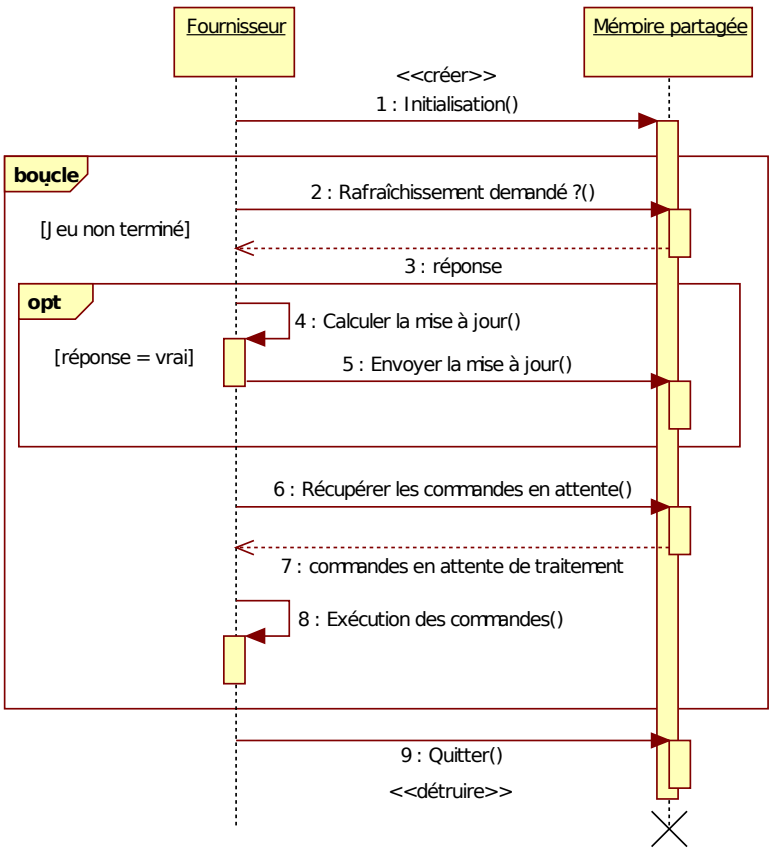



FIGURE 4.12 – Gestion de la mémoire partagée par le « Fournisseur ».

Dans un premier temps, le « Fournisseur » crée et initialise la mémoire partagée de manière à ce qu'elle soit accessible par le « Client ». Après cette initialisation, le jeu entre dans la boucle de simulation et à chaque itération le « Fournisseur » consulte la mémoire partagée pour savoir si un rafraîchissement est demandé. Si tel est le cas, le « Fournisseur » calcule la mise à jour en fonction de l'état courant du jeu et la copie dans la mémoire partagée. Le « Fournisseur » s'attache ensuite à traiter les commandes définies par le « Client ». Enfin lorsque le jeu est terminé, le « Fournisseur » nettoie et libère la mémoire partagée.

Du côté « Client », l'appel aux fonctions de l'API peut être réalisé après s'être connecté à la mémoire partagée et avoir demandé un premier rafraîchissement. Dans le cas contraire, les fonc-

tions de l'API retournent un code d'erreur ou lèvent une exception en fonction du langage utilisé. Lors d'une demande de rafraîchissement, cet appel est bloquant tant que le « Fournisseur » n'a pas détecté et réalisé la mise à jour. La figure 4.13 illustre une séquence d'interaction entre le « Fournisseur », le « Client » et la mémoire partagée avec le cas particulier d'une demande de rafraîchissement.

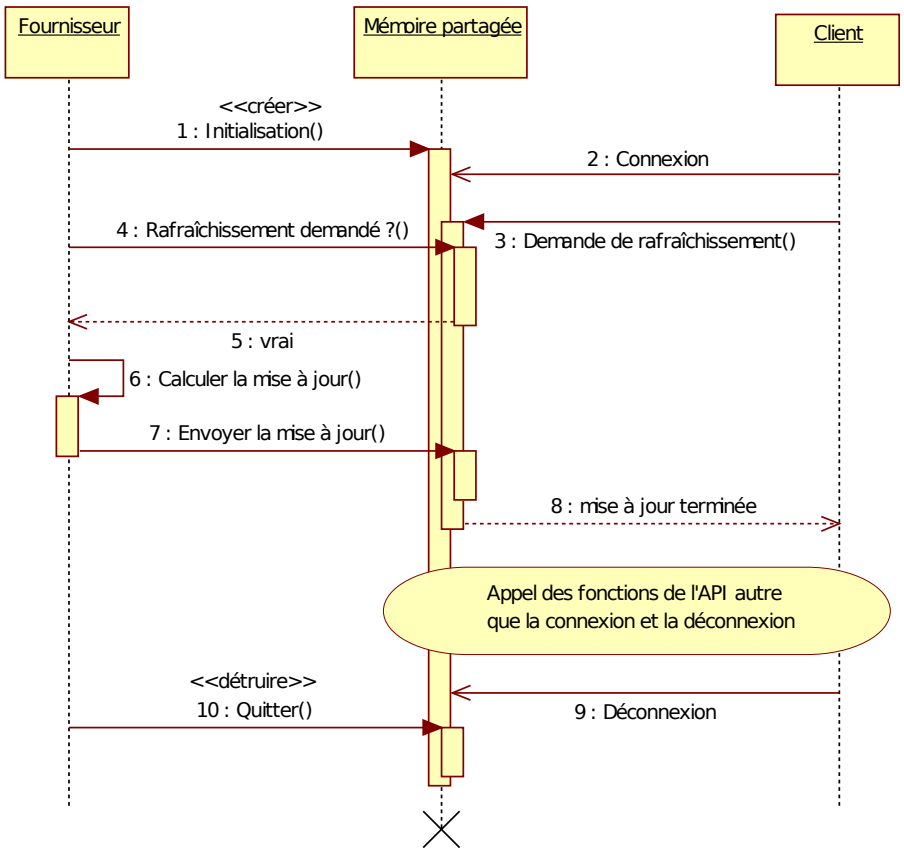



FIGURE 4.13 – Cas particulier du rafraîchissement côté « Client ».

### Description détaillée

Le diagramme de classes UML de la figure 4.14 décrit les structures internes à l'API ainsi que les différentes relations entre celles-ci.

Il convient d'apporter quelques explications sur ce diagramme en commençant par la structure qui représente l'ensemble des données transférables via la mémoire partagée : *PP_Shared-*

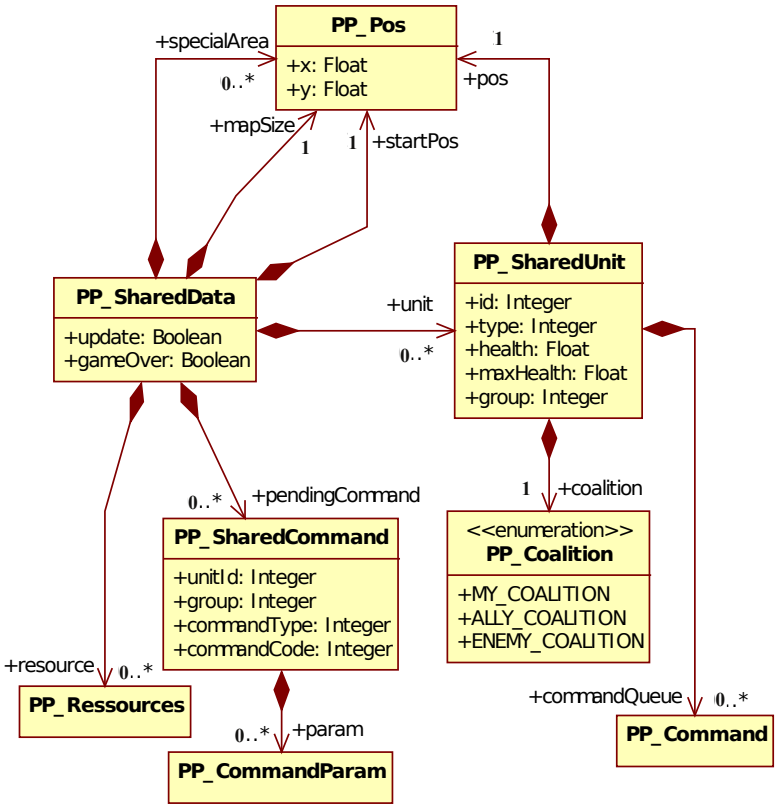



FIGURE 4.14 – Diagramme de classes de l'API Prog&Play.

*Data*. Elle contient notamment l'ensemble des unités accessibles par l'interface « Client ». Chaque unité est représentée par la structure *PP_SharedUnit* qui contient un identifiant, un type, un capital santé courant et maximum, un groupe, une position, un ensemble de commandes à traiter et une coalition. Cette dernière est représentée par l'énumération *PP_Coalition* qui est composée de trois membres : *MY_COALITION* représente les unités contrôlées par le joueur ; *ALLY_COALITION* représente les unités contrôlées par un allié du joueur ; *ENEMY_COALITION* représente les unités contrôlées par un ennemi du joueur.

La structure *PP_SharedData* contient également une liste de zones spéciales (*specialArea*) qui permet de transférer un ensemble de positions via l'API. Il importe alors en fonction du jeu et de ses données de définir la représentation de ces positions de façon à ce qu'elles soient utilisables par le joueur.

Une dernière précision concerne la représentation des commandes qui intervient sous la forme de deux structures : *PP_Command* et *PP_SharedCommand*. La première représente l'ensemble des commandes à traiter pour une unité. Ces commandes ont été définies par le joueur ou son programme lors des pas de simulation précédents. La seconde, (*PP_SharedCommand*), représente l'ensemble des commandes définies par le programme du joueur et non encore traitées par le jeu. Si les instances de cette structure sont valides, les commandes associées se retrouveront sous la forme d'instances de *PP_Command* lors des mises à jour à venir.

Le diagramme de composants UML de la figure 4.15 décrit l'organisation du système du point de vue des éléments logiciels et exprime donc les dépendances entre le programme du joueur et le jeu. Les deux interfaces « Fournisseur » et « Client » sont disponibles via une implémentation en C détaillée en annexe A. Ce langage a été choisi comme base de l'API en raison de son utilisation répandue qui en fait un langage interopérable avec de nombreux autres langages de programmation.

Pour implémenter le système Prog&Play, le choix s'est orienté vers l'utilisation de la bibliothèque *Boost interprocess*[49] qui propose une solution portable et efficace de gestion des mémoires partagées. D'autre part, cette bibliothèque offre la possibilité d'utiliser dans la mémoire partagée des conteneurs complexes comme les vecteurs ou les maps par exemple.

## Conclusion

L'API Prog&Play a été conçue pour cacher la complexité de synchronisation d'une communication interprocessus et donner la possibilité d'accéder à un sous ensemble des données du jeu. Ce sous ensemble de données permet de simplifier la compréhension du jeu en réduisant la quantité d'informations accessibles par les programmes des étudiants. Cette simplification a un second intérêt lié à la portabilité de l'API Prog&Play, car, si cette API était trop proche des spécifications d'un jeu particulier, il serait alors difficile, voire même impossible, de l'utiliser avec de nouveaux jeux. Cette simplification des données implique des limitations quant à la réalisation des IA possibles. En effet, en simplifiant les données accessibles, certaines informations de l'état du jeu et certaines fonctionnalités ne sont plus accessibles à travers l'API. Le choix des données transférées à travers l'API Prog&Play est un point critique toujours en cours d'évolution. Actuellement, un joueur a accès, à travers l'API, à l'état de fin de partie, à la taille

49. `http://www.boost.org/doc/libs/1_39_0/doc/html/interprocess.html` consulté le 8 Janvier 2010

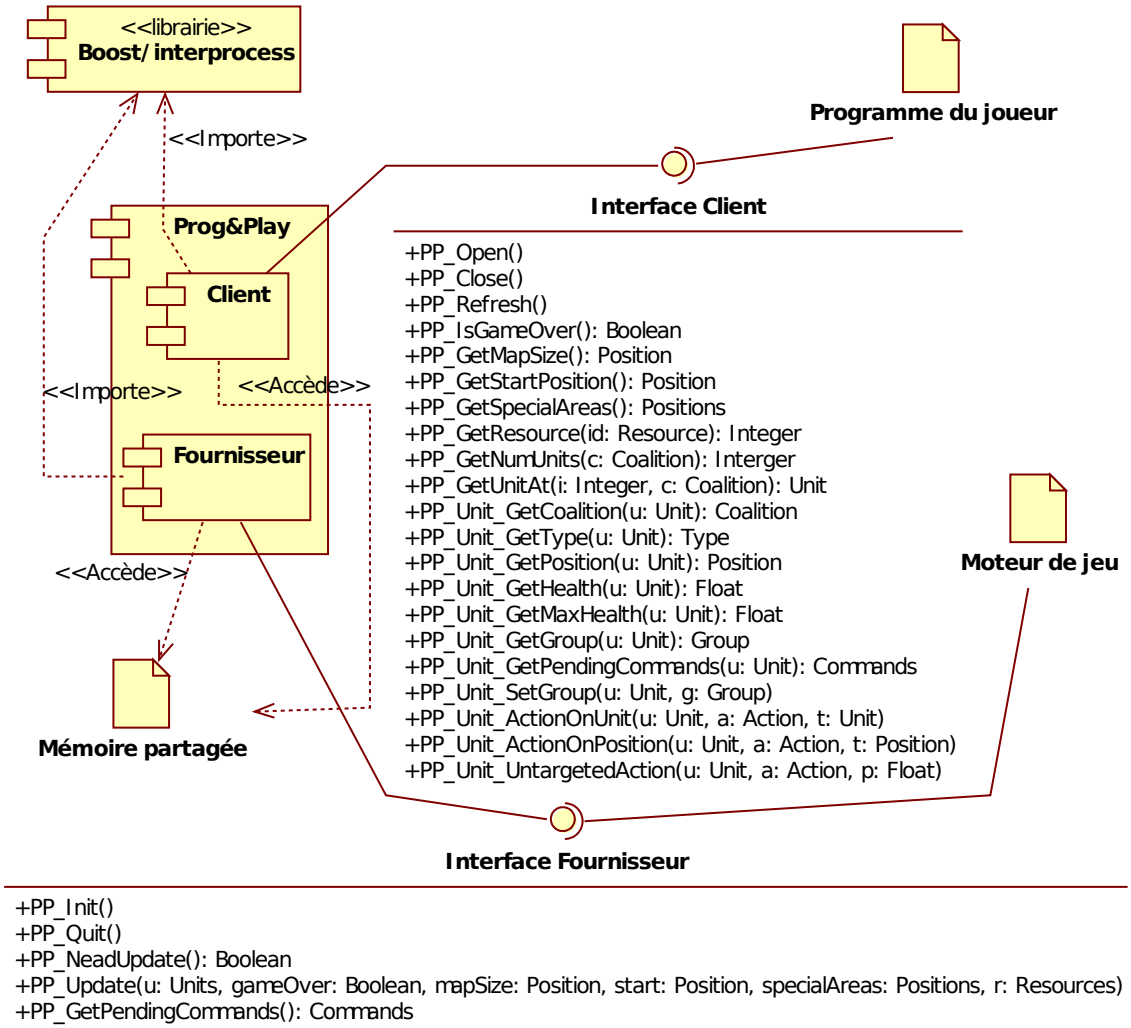



FIGURE 4.15 – Diagramme de composants du système Prog&Play.

de la carte, à sa position de départ sur la carte, aux unités visibles (non cachées par le brouillard de guerre), à ses réserves de ressources et à un ensemble de positions particulières.

L'utilisabilité du système Prog&Play a été vérifiée au fur et à mesure de son développement à travers son intégration dans différents moteurs de STR tels que ORTS, *Spring* et *WarZone 2100*.

## 4.3 Intégration de Prog&Play dans différents jeux de STR

Intégrer le système Prog&Play dans un jeu de STR requiert d'y apporter des modifications et donc d'avoir accès au code source. La principale difficulté liée à cette tâche consiste à comprendre le code du jeu afin de déterminer dans quel module intégrer les modifications. En effet, les moteurs de jeux de STR dans lesquels l'API Prog&Play a été intégrée sont des applications complexes de plusieurs dizaines de milliers de lignes de code. Cette complexité confirme la nécessité d'une interface adaptée aux compétences et connaissances des étudiants novices en programmation.

L'intégration du système Prog&Play dans plusieurs jeux de STR laisse le choix aux enseignants du jeu à utiliser. En effet, l'univers du jeu et les caractéristiques de *gameplay* peuvent influencer la mise en place des exercices de programmation et donc être plus ou moins adaptés au profil de leurs étudiants.

La première version du système Prog&Play a été développée sur ORTS. La seconde version, quant à elle, a été intégrée à *Spring* et *WarZone 2100*.

### 4.3.1 Intégration à ORTS

ORTS est un moteur de jeu de STR atypique du point de vue de son architecture réseau client-serveur. L'intégration de cette version de Prog&Play, basée sur l'utilisation d'une bibliothèque dynamique, est réalisée par la modification du logiciel client. Ces modifications consistent à intégrer le CDD ainsi que les composants nécessaires au chargement de la bibliothèque dynamique tels que le « Chargeur » et l'« IMJ ».

La mise en œuvre du « Chargeur » dans le client d'ORTS consiste à implémenter l'algorithme 1 présenté page 88 avec notamment le passage de l'« IMJ » à la bibliothèque dynamique en appelant la fonction `IGCU_Start`. Dans le contexte d'ORTS, l'« IMJ » est composée de deux points d'entrée. Le premier donne accès à l'état du jeu (le *gsm* ou *GameStateModule*) et le second donne accès au contrôle des unités (le *pc* ou *PlayerCommander*). À chaque pas de simulation, le serveur envoie à chaque client sa propre vue. De cette manière, la tricherie est impossible car les données accessibles via le *gsm* ont été validées par le serveur. Par conséquent, seuls les objets visibles par le joueur sont accessibles via le *gsm*. Le *pc*, quant à lui, fournit un ensemble de méthodes utiles au contrôle des unités, comme définir un déplacement, une attaque ou une construction. À noter que le *gsm* et le *pc* sont des instances de classes internes à ORTS.

Elles donnent un accès complet aux données mais restent complexes à utiliser, en particulier pour des débutants en programmation.

Pour intégrer la seconde version de Prog&Play, le choix du jeu support s'est orienté vers le moteur de *Spring*. La raison de ce changement de moteur est due à la richesse et la plus grande stabilité de *Spring* par rapport à ORTS qui en est à un stade plus expérimental.

### 4.3.2 Intégration à *Spring*

Spring, fort d'une communauté active, présente de nombreux jeux et ressources associées. Intégrer Prog&Play à Spring rend l'API automatiquement compatible à une gamme importante de jeux de STR et augmente donc par conséquent l'éventail des jeux sérieux réalisables.

*Spring* est basé sur une architecture répartie de type P2P. Dans cette architecture, la base de données représentant le monde virtuel est dupliquée sur les différentes machines. Par conséquent, chaque partie prenante du jeu accède à l'ensemble des données telles que les positions ou le nombre d'unités adverses. Le « brouillard de guerre » joue alors son rôle de filtre pour cacher les données non accessibles au joueur en fonction de la position et du champ de vision de ces unités. Dans Prog&Play, les données accessibles via l'API « Client » doivent respecter la visualisation graphique telle qu'elle est définie au moment d'un rafraîchissement. Lors de l'intégration de la partie « Fournisseur » de Prog&Play dans le jeu, une attention particulière doit donc être portée sur le filtrage des données transférées dans la mémoire partagée. À titre d'exemple, pour déterminer si une unité est accessible via la partie « Client » de Prog&Play, elle doit soit appartenir au joueur ou à l'un de ses alliés, soit appartenir à un ennemi et être visible par le joueur. Lorsqu'une unité est sélectionnée pour être accessible à travers l'interface « Client », l'ensemble de ces attributs compatibles avec la structure de données de Prog&Play doit être extrait et copié dans la mémoire partagée (voir la structure « *PP_SharedUnit* » de la figure 4.14 page 95).

La structure « *PP_SharedData* », interne à l'API Prog&Play (voir figure 4.14 page 95), contient une liste de zones spéciales (*specialArea*) prévue pour stocker un ensemble de positions. Dans *Spring*, cette liste est utilisée pour transférer les emplacements des sources géothermiques. Ces dernières sont utilisées de deux manières différentes par les jeux construits sur *Spring*. La majorité utilise les sources géothermiques comme une ressource utilisable par des centrales pour générer de l'électricité. *Kernel Panic*, quant à lui, les redéfinit comme des sources de données qui représentent les emplacements où la construction de structures est possible.

L'API Prog&Play est conçue pour être intégrée à plusieurs jeux de STR. Cependant, pour utiliser la partie « Client », quelques informations doit être fourni en complément de la spécification du langage de programmation utilisé. En effet, les types d'unités et les actions réalisables sont propres à chaque jeu de STR. Par conséquent, pour interpréter le type d'une unité ou pour définir une action à réaliser, un ensemble de constantes doivent être fournies au joueur en même temps que la spécification de la partie « Client ». À titre d'exemple dans KP, l'ordre de déplacement (*MOVE*) est représenté par la valeur « 10 », l'ordre de construction d'un SOCKET par la valeur « -41 » et le type des unités BIT et OCTET sont respectivement « 4 » et « 7 ». La définition d'un ensemble de constantes pour identifier chaque type d'unité et chaque action est un complément indispensable à l'utilisation de la partie « Client » de Prog&Play.

Moyennant la génération d'une liste de constantes pour chaque jeu basé sur *Spring*, l'API Prog&Play est compatible avec chacun d'entre eux. La seconde version de Prog&Play a également été intégrée dans un deuxième moteur de jeu de STR : *WarZone 2100*.

### 4.3.3 Intégration à *WarZone 2100*

L'intégration de la seconde version de Prog&Play à *WarZone 2100* poursuit deux objectifs : montrer que l'API est compatible avec un produit originairement commercial et augmenter le nombre de jeux compatibles Prog&Play et en conséquence l'espace des jeux sérieux réalisables.

À l'image de *Spring*, *WarZone 2100* est basé sur une architecture P2P qui nécessite de porter une attention particulière aux données transférées dans la mémoire partagée. L'enjeu étant de fournir une information exacte afin d'éviter l'accès à des données cachées et donc la tricherie.

Pour *WarZone 2100*, les zones spéciales sont utilisées pour stocker la position des puits de pétroles sur lesquels peuvent être construit des derricks. Le pétrole est la seule ressource de ce jeu, elle est nécessaire à la construction des bâtiments et des unités et à la recherche de nouvelles technologies.

La particularité de *WarZone 2100* porte sur un système de conception d'unités qui permet au joueur à partir d'un châssis, d'un système de propulsion et d'une tourelle de concevoir les unités adaptées à sa stratégie de jeu. Cette fonctionnalité a pour conséquence de complexifier la génération des constantes nécessaires à l'utilisation de l'interface « Client ». En effet, avec plus de 400 technologies différentes, le nombre de constantes nécessaires pour caractériser toutes les unités est tel quelles en deviennent inutilisables. En se cantonnant aux unités mobiles, les 77 technologies de tourelles combinées aux 14 technologies de châssis et aux 5 technologies

de propulsions permettent de réaliser 5390 unités différentes. Pour pallier à cette combinatoire importante et faciliter l'utilisation de l'API, un type d'unité représente une famille qui regroupe plusieurs sortes d'unités. À titre d'exemple, les familles « Constructeur » et « Transporteur », qui regroupent de nombreux types d'unités différentes, sont respectivement représentés par les valeurs « 3 » et « 6 ». En conséquence, la conception et la création d'unités dans *WarZone 2100* ne peuvent être pilotées à travers l'API Prog&Play. En revanche, les actions plus classiques comme un déplacement ou la construction d'un bâtiment sont réalisables. Pour illustration, l'ordre de déplacement (*MOVE*) est représenté par la valeur « 2 » et l'ordre de construction d'une usine de niveau 1 est représenté par la valeur « -851971 ».

Cette originalité de *WarZone 2100* illustre la limite de l'API Prog&Play, qui, en raison de la généralité des données manipulées lui permet d'être intégrée dans des jeux de STR différents mais ne peut retranscrire les subtilités propres à chacun de ces jeux. Toutefois, cette limite renforce l'une des spécifications du jeu sérieux : maintenir l'interaction classique entre le joueur et le jeu. De cette manière, au cours de la simulation, le joueur est actif (il joue) pour réaliser les tâches non prises en compte par son programme.

## Conclusion

Pour illustrer concrètement la portabilité de la deuxième version de l'API Prog&Play dans différents jeux de STR, le programme en langage C de la figure 4.16 fonctionne sur *WarZone 2100* et l'ensemble des jeux de Spring (*XTA*, *Balanced Annihilation*, *Star Wars : Imperial Winter* et *Kernel Panic* entre autres).

L'intégration des deux versions de Prog&Play a donc été réalisée dans plusieurs moteurs de STR, l'objectif étant de tester l'utilisabilité du système dans différents contextes de jeu. La principale difficulté rencontrée porte sur la compréhension du code des moteurs étudiés qui manquent parfois de documentation détaillée. Cette difficulté se trouverait grandement réduite si l'intégration était réalisée par un ou plusieurs développeurs du jeu qui ont une connaissance et une maîtrise du code de leur projet bien plus importante que la notre.

Concernant la seconde version, l'intégration à *Spring* et à *WarZone 2100* a montré la nécessité de fournir une spécification des constantes utiles pour représenter les données du jeu. L'annexe B présente une implémentation en langage C de l'ensemble des constantes utiles à la gestion des unités de la faction « Système » de *Kernel Panic*.

```c
#include "PP_Client.h" /* interface "Client" de l'API Prog&Play */
#ifdef SPRING
/* Liste des constantes propres a Spring */
#include "constantsList_Spring.h"
#else
/* Liste des constantes propres a WarZone 2100 */
#include "constantsList_WZ.h"
#endif

int main (void){
    int i;
    /* definition de la position cible */
    PP_Pos p;
    p.x = 10.0;
    p.y = 10.0;
    PP_Open(); /* ouverture de l'API Prog&Play */
    PP_Refresh(); /* Demande d'une mise a jour des donnees */
    /* Parcours de toutes les unites */
    for (i = 0 ; i < PP_GetNumUnits (MY_COALITION) ; i++){
        /* Ordonner a l'unite courante de se deplacer */
        PP_Unit_ActionOnPosition(PP_GetUnitAt(MY_COALITION, i), MOVE, p);
    }
    PP_Close(); /* Fermeture de l'API Prog&Play */
    return 0;
}
```

FIGURE 4.16 – Exemple de programme en langage C fonctionnel avec plusieurs jeux de STR intégrant le système Prog&Play.
*Cet exemple déplace l'ensemble des unités du joueur à la position* $(10, 10)$.

Les différentes versions de Prog&Play et leurs intégrations dans plusieurs moteurs de STR montrent la possibilité d'utiliser ces moteurs comme bases à la conception de jeux sérieux pour l'apprentissage de la programmation.

## 4.4 Conception du jeu sérieux

À ce stade, les jeux de STR associés à Prog&Play sont des outils pour pratiquer la programmation mais ne constituent pas encore des jeux sérieux. En effet, la dernière étape manquante consiste à motiver le joueur à pratiquer la programmation dans ce type d'environnement en vue de lui faire atteindre des objectifs d'apprentissage. Une première tentative de conception de jeu

sérieux a été nécessaire avant d'en réaliser un suffisamment structuré pour être déployé dans un contexte réel d'enseignement.

### 4.4.1 Première version de Prog&Play avec ORTS

Cette première expérimentation a consisté à développer une application en lien direct avec un cas d'enseignement réel. Les étudiants de l'université Paul Sabatier (Toulouse III) inscrits en première année de licence STS (Science, Technologies, Santé) qui suivent au second semestre la majeure IMM (Informatique, Mathématiques, Mécanique) ont dans leur enseignement de l'informatique un mini projet à réaliser durant les dernières séances de travaux pratiques (TP). Le sujet de l'année universitaire 2007-2008 portait sur la réalisation en langage C d'un résolveur de sudoku. Au cours des cinq dernières séances, les étudiants étaient guidés dans la réalisation de ce résolveur. Ils étaient, à cette occasion, confrontés à la manipulation des entrées/sorties, des tableaux, des types de données, des sous-programmes et des techniques de résolution.

L'objectif fixé consistait à vérifier la possibilité de transposer un exercice de TP dans le contexte d'ORTS. À ce titre, une carte de jeu spéciale a été créée pour représenter la grille du sudoku (voir figure 4.17) ainsi qu'un ensemble de vaisseaux représentant les 81 chiffres positionnables sur le sudoku (voir figure 4.18).

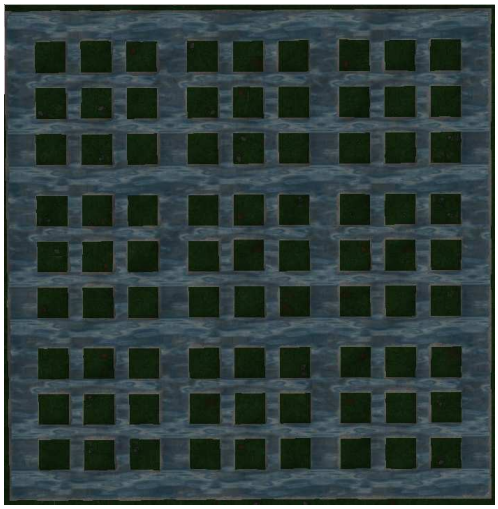


FIGURE 4.17 – Grille de sudoku vide dans ORTS.

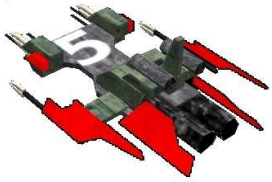



FIGURE 4.18 – Un vaisseau représentant un chiffre à positionner sur la grille de sudoku.

Pour simplifier la manipulation des vaisseaux une interface de haut niveau est proposée. Elle permet d'abstraire la complexité du *gsm* et du *pc* et se compose seulement de deux procédures :

- void **ORTS_initialise_jeux**(void) : initialise un ensemble de données permettant de gérer le positionnement des vaisseaux et se charge, en outre, de déplacer les vaisseaux à l'extérieur du sudoku s'il en reste dans la grille.
- void **ORTS_positionne_vaisseau**(int *l*, int *c*, int *candidat*) : cherche un vaisseau de numéro « *candidat* » (inclus dans l'intervalle $[1; 9]$) à l'extérieur du sudoku et le positionne sur la case située sur la ligne numéro « *l* » (incluse dans l'intervalle $[0; 8]$) et la colonne numéro « *c* » (incluse dans l'intervalle $[0; 8]$).

Grâce à ces deux fonctions, il est possible de positionner les vaisseaux pour représenter l'état initial de la grille de sudoku (voir figure 4.19) et d'afficher la solution générée par le résolveur dans le contexte du jeu (voir figure 4.20).

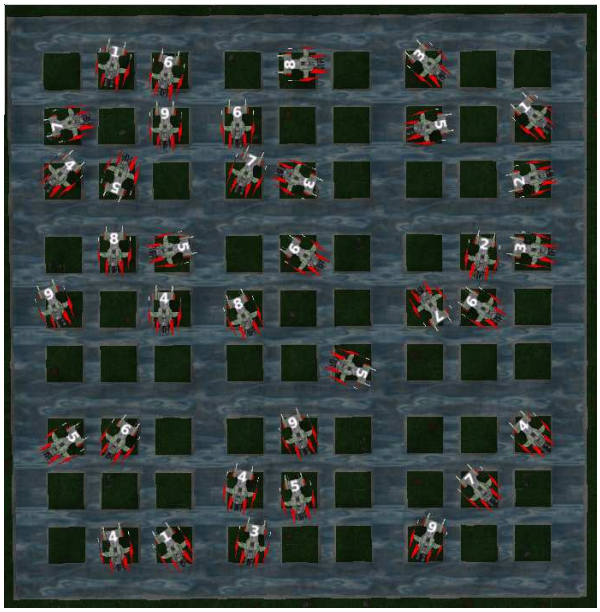


FIGURE 4.19 – Vaisseaux positionnés conformément à l'état initial de la grille de sudoku (avant résolution).

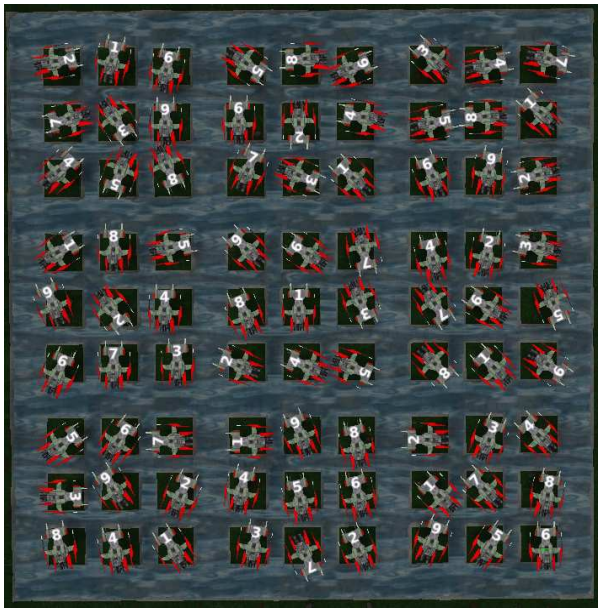


FIGURE 4.20 – Vaisseaux positionnés conformément à la solution de la grille de sudoku (après résolution).

Cette application est donc fonctionnelle, elle illustre la possibilité de projeter des exercices de programmation réels dans le contexte d'ORTS. Toutefois, elle ne constitue pas un jeu sérieux de STR au sens strict car la résolution d'un sudoku et le positionnement de vaisseaux sur une carte n'a aucun lien avec le principe, les règles et les buts des jeux de STR. Cet exemple

montre toute la difficulté de conception d'un jeu sérieux. Il ne suffit pas d'habiller un problème quelconque avec un environnement 3D pour prétendre réaliser un jeu sérieux, car, dans ce cas, la composante « jeu » est inexistante et la motivation espérée en conséquence s'en trouve fortement réduite. Pour cette raison, cette application n'a pas été testée avec des étudiants en contexte réel. En effet, l'évaluation doit porter sur un jeu sérieux où la programmation est une composante importante du *gameplay* de manière à exploiter les caractéristiques des jeux de STR et à bénéficier de leur potentiel motivationnel.

### 4.4.2 Seconde version de Prog&Play avec *Spring*

Pour réaliser la version aboutie, *Spring* s'est avéré plus performant qu'ORTS en raison de la richesse des jeux disponibles sur ce moteur. Le choix s'est orienté vers le jeu *Kernel Panic* en raison des trois caractéristiques suivantes :

- la simplicité de l'arbre des technologies et l'inexistence de la gestion de ressources permettent aux joueurs novices de comprendre et maîtriser rapidement les règles et buts du jeu ;
- le jeu appartient au domaine public ce qui permet une utilisation totalement libre et gratuite ;
- l'univers informatique du jeu est en lien direct avec le contenu de la discipline à enseigner.

Pour profiter de la motivation induite par les jeux de STR, l'intégration des objectifs d'apprentissage doit se mêler aux structures de jeu préexistantes. Le mode campagne et le mode escarmouche sont deux formes de jeu différentes à exploiter pour la réalisation du jeu sérieux.

#### Le mode campagne

Le mode campagne est structuré autour d'un scénario divisé en missions. Chacune d'elles propose au joueur d'atteindre un objectif afin de passer à la suivante et ainsi progresser dans le scénario. La motivation, dans ce mode de jeu, est maintenue par la volonté du joueur d'être acteur dans la résolution de l'intrigue.

Ce mode de jeu peut être utilisé dans une approche classique de l'enseignement basé sur la confrontation de l'étudiant à une série d'exercices. Chaque mission pose un problème que l'étudiant doit résoudre à l'aide d'un programme informatique. Toute la difficulté consiste alors à proposer des problèmes qui mettent en exergue le savoir à enseigner et qui soient liés au scénario du jeu. Dans le contexte de *Kernel Panic* et des objectifs d'apprentissage centrés sur les fonda-

mentaux de la programmation, le scénario de la campagne proposée est le suivant : « *Depuis un certain nombre d'années, une guerre secrète fait rage au sein même des ordinateurs. Des attaques ont régulièrement lieu contre d'innocentes victimes. Aujourd'hui c'est votre tour. Votre agresseur a capturé le contrôleur de votre souris. Vous devez le récupérer. Votre seule solution : la programmation* ». Cette campagne est actuellement divisée en sept missions :

- **Mission 1** : « *Lors de la précédente attaque, vous avez perdu de nombreuses unités et celles qui vous restent sont dispersées sur la carte. Un seul BIT est encore sous votre contrôle. Un rapport vous est parvenu vous indiquant qu'un OCTET se trouverait au point de ralliement (1983, 1279). Déplacez votre unique entité à cette position pour tenter de retrouver l'OCTET perdu. Bon courage commandant...* ». Dans cette mission, le joueur contrôle une seule unité. Il doit simplement la déplacer à la position indiquée dans le briefing. Pour atteindre cet objectif, le joueur doit réaliser un petit programme en manipulant des variables, des types, des affectations, des fonctions et le passage de paramètre. Lorsque le BIT atteint la position indiquée, l'OCTET recherché ne s'y trouve pas. La deuxième mission est alors proposée ;
- **Mission 2** : « *Le rapport que vous aviez reçu était correct mais l'OCTET ne se trouve plus au point de ralliement prévu. Des traces vous indiquent qu'il s'est éloigné dans la direction 119° par rapport au Nord (sens anti-horaire). La fraîcheur des indices vous indique qu'il n'est pas loin. Déplacez votre BIT de 1060 unités dans cette direction* ». Cette mission reprend les concepts de programmation de la première mission et ajoute un calcul trigonométrique pour déterminer la position à atteindre en fonction d'un angle et d'une distance à parcourir. Elle illustre par ce fait la possibilité d'intégrer des calculs mathématiques dans le jeu en lien avec le scénario. À la fin de cette mission, lorsque le BIT a atteint la position correctement calculée, l'OCTET recherché est découvert et la troisième mission est débloquée.
- **Mission 3** : « *L'OCTET, qui vient de se rallier à vous, vous informe que d'autres entités se sont regroupées non loin de là. Il vous communique les coordonnées de la formation d'OCTETS qu'il tentait de rejoindre avant votre arrivée (479, 1825). Il vous indique également qu'un ensemble de BITS s'est rassemblé aux coordonnées (1400, 1371). Afin de récupérer ces nouvelles forces, ordonnez à vos deux entités de rejoindre leurs compatriotes respectifs* ». Dans cette mission le joueur contrôle deux unités et doit donner un ordre de déplacement différent en fonction de leur type (BIT ou OCTET). Tout naturellement la structure de contrôle conditionnelle est introduite pour opérer cette sélection.

- **Mission 4** : « *Toutes les entités que vous possédez ont subi de lourds dommages lors de la précédente attaque. Vous devez les réparer avant de lancer la contre-attaque. Le dernier ASSEMBLEUR encore en marche et capable d'effectuer les réparations est en route vers le point de ralliement aux coordonnées (256, 1024). Déplacez toute votre armée à la rencontre de cet ASSEMBLEUR* ». Dans cette mission le joueur contrôle l'ensemble des unités récupérées à la mission 3 et doit toutes les déplacer vers une position précise. Une structure de contrôle itérative doit donc être utilisée pour parcourir l'ensemble des unités et ordonner à chacune d'entre elles de se déplacer vers le point de ralliement indiqué.
- **Mission 5** : « *L'ASSEMBLEUR vient d'apparaître sur la carte, aidez le à rejoindre le reste de votre armée. Déplacez uniquement l'ASSEMBLEUR aux coordonnées (256, 811)* ». Dans cette mission, le joueur contrôle un ensemble d'unités composé des entités de la mission 4 et de l'ASSEMBLEUR. Pour déplacer uniquement ce dernier, le joueur doit le rechercher parmi la liste des unités contrôlées. L'utilisation de structures de contrôle conditionnelles et itératives est requise pour résoudre ce problème.
- **Mission 6** : « *Votre armée est maintenant regroupée et vous disposez du dernier ASSEMBLEUR du secteur encore en marche. Ordonnez-lui de réparer toute votre armée* ». Pour réaliser cette réparation, l'imbrication de structures de contrôle conditionnelles et itératives est nécessaire. Une difficulté supplémentaire est introduite à travers l'utilisation de la fonction « `PP_Refresh` » qui permet d'obtenir les mises à jour des données du jeu et ainsi suivre l'état de réparation des unités. Lorsque toutes les unités sont réparées, la dernière mission de la campagne est débloquée.
- **Mission 7** : « *Votre armée est fin prête à retourner au combat. Le contrôleur de votre souris est actuellement aux mains de votre adversaire. Il est temps d'aller le récupérer. Lancez l'attaque sur la position (1792, 256). N'oubliez pas que l'ASSEMBLEUR peut vous être d'une aide précieuse... Bonne chance commandant* ». Cette dernière mission, volontairement ouverte, est déséquilibrée en faveur de l'ordinateur. Aucune indication n'est donnée sur la démarche à suivre pour remporter la victoire. Simplement, la dernière phrase du briefing a pour objectif d'inciter les étudiants à intégrer l'ASSEMBLEUR dans leur stratégie. Ils peuvent s'en servir pour rétablir l'équilibre en réparant les unités endommagées en cours de combat et en construisant des structures afin de produire des unités supplémentaires.

En résumé, le joueur commence la campagne avec un seul BIT et doit s'en servir pour récupérer des unités dispersées sur la carte. Une fois son armée reconstituée et prête au combat,

le joueur peut mener une attaque pour récupérer le contrôleur de la souris et recouvrer son utilisation. Une solution en langage C des six premières missions est proposée en annexe C. La septième étant plus ouverte ne présente pas de solution type.

Grâce à ce jeu de missions, le joueur est acteur du déroulement de l'histoire et progresse dans le scénario après chaque objectif atteint. Cette dynamique est utilisée pour introduire progressivement dans chaque mission un concept algorithmique ou une difficulté supplémentaire en vue de proposer de nouveaux défis en lien avec le savoir à enseigner. La séquence est donc introduite lors de la première mission, la structure de contrôle conditionnelle lors de la troisième, la structure de contrôle itérative lors de la quatrième et les structures de contrôle imbriquées lors de la sixième. La dernière mission, plus ouverte, permet aux étudiants de concevoir et d'implémenter une stratégie de jeu à l'aide des compétences acquises lors des six premières missions.

Dans ce scénario une contrainte forte est imposée, le contrôle du jeu à l'aide de la souris est désactivé. La seule méthode d'interaction possible reste donc la programmation. Cette campagne présente donc un double objectif : véhiculer les fondamentaux de la programmation et former les étudiants à l'utilisation de l'API pour qu'ils puissent la réutiliser dans un contexte plus ouvert. Le mode escarmouche peut en être la suite logique.

#### Le mode escarmouche

Le mode escarmouche est basé sur un modèle de compétition. Le joueur avec ses éventuels alliés doit atteindre un objectif et se confronter à un ou plusieurs adversaires. La motivation dans ce mode de jeu est maintenue par la volonté du joueur à atteindre une performance supérieure à ses adversaires.

L'exploitation de ce mode de jeu dans l'enseignement repose sur l'utilisation d'une approche par projet qui consiste à laisser les étudiants concevoir et réaliser leurs propres IA. L'objectif final est de permettre aux étudiants d'utiliser leurs productions dans une compétition organisée. Plusieurs modalités de mise en œuvre sont alors définies notamment vis-à-vis des projets proposés et de leur encadrement. Concernant les compétitions, deux solutions sont envisageables.

**Première solution :** utiliser le mode multijoueur « classique » de *Kernel Panic*. Chaque joueur choisit sa faction « Système », « Pirate » ou « Réseaux » et commence la partie avec l'unité maîtresse associée. À partir de celle-ci, le joueur génère des unités et développe sa stratégie. Dans le contexte de Prog&Play, la seule différence consiste à autoriser l'utilisation de pro-

grammes informatiques réalisés avec l'API. L'objectif, ici, est de faire réaliser aux étudiants un programme utilisable lors de jeux multijoueurs. Ils sont donc amenés à analyser leur stratégie de jeu et à imaginer les parties qui peuvent être automatisées et déléguées à un programme. Voici quelques exemples d'algorithmes :

- **espionnage** : l'objectif de cet algorithme consiste à contrôler un groupe d'unités chargé d'explorer la carte en vue de surveiller les positions de l'adversaire. Connaître les mouvements de l'adversaire permet d'anticiper ses actions et d'adapter sa stratégie en conséquence. Sans la programmation, cette activité est fastidieuse car le joueur doit manuellement déplacer ses espions, être attentif à leurs découvertes et gérer le renouvellement du groupe. Le temps passé à ces tâches ne peut être consacré aux autres activités de la stratégie.
- **miner le terrain** : l'objectif de cet algorithme consiste à miner des points stratégiques de la carte de jeu en vue de ralentir l'expansion des adversaires. Bien souvent dans *Kernel Panic* le vainqueur est le joueur qui a su prendre rapidement des positions et empêcher ses adversaires de se développer. Comme l'espionnage, la gestion d'un champ de mines demande au joueur beaucoup de temps pour reconstruire les mines détruites et soutenir les percées éventuelles des adversaires. L'utilisation d'un programme qui prend en charge cette partie peut fournir un avantage conséquent au joueur.
- **réparer les unités endommagées** : cet algorithme est voué à être utilisé dans les phases de combat pour soutenir les unités engagées. L'objectif consiste à utiliser les capacités de certaines unités (comme l'ASSEMBLEUR) pour maintenir les unités stratégiques en bonne santé. L'issue d'une confrontation entre deux groupes d'unités peut basculer en faveur du plus faible si celui-ci est soutenu par une maintenance efficace. Un joueur qui a réalisé le mode campagne pourra réutiliser le code de la sixième mission et l'adapter au contexte de jeu multijoueur.
- **repli automatique** : à l'image de l'algorithme précédent, celui-ci peut être utile au cours d'une confrontation. Il s'attache à estimer l'issue d'un combat en vue de déterminer s'il semble judicieux de le poursuivre. En effet, générer de nouvelles unités prend du temps, il convient alors de ne pas en perdre inutilement. Une estimation juste des risques encourus permet d'éviter les pertes inutiles et fait donc gagner du temps au joueur concerné.

**Seconde solution :** définir un cadre plus restreint où toutes les fonctionnalités du jeu ne sont pas accessibles. Voici trois exemples de compétitions inspirées de celles organisées autour du jeu ORTS :

1. **exploitation de ressources** : cette première catégorie ne peut mise en œuvre avec *Kernel Panic* car elle est basée sur l'exploitation des ressources inexistantes dans ce jeu. En revanche, *Balanced Annihilation*, un des autres jeux de *Spring* compatible Prog&Play, peut être utilisé. Chaque joueur commence la partie avec une unité maîtresse (le *Commander*) positionnée aléatoirement sur la carte. Le monde est peuplé d'unités neutres et invincibles qui se déplacent aléatoirement sur la carte. Seules les branches de « l'arbre des technologies » associées à la gestion des ressources sont activées si bien que les seules constructions possibles sont les structures de production, de stockage d'énergie et de métal ainsi que les unités de constructions de ces structures. L'objectif est de mettre en place une chaîne de production d'énergie et de métal la plus productive possible en un temps imparti.
2. **combat stratégique** : cette deuxième catégorie est parfaitement applicable à *Kernel Panic*. Deux joueurs commencent la partie avec cinq SOCKETS chacun et dix OCTETS positionnés autour de chaque SOCKETS. Les SOCKETS du premier joueur sont positionnés aléatoirement sur la carte. Ceux du second joueur sont placés de manière symétrique. Le brouillard de guerre est désactivé et à l'image de la première catégorie, le monde est peuplé d'unités neutres et invincibles qui se déplacent aléatoirement sur la carte. L'objectif de ce jeu est de détruire autant de SOCKETS adverses que possible en un temps imparti.
3. **combat à petite échelle** : cette troisième catégorie se rapproche des batailles de robots présentées dans la section 3.2.2 page 69. Deux joueurs commencent la partie avec une cinquantaine de BITS chacun positionnés aléatoirement dans les premiers quarts gauche et droit de la carte. Cette dernière est plane et le brouillard de guerre est désactivé. L'objectif de ce jeu est de détruire autant d'unités adverses que possible en un temps imparti.

Quelque soit la mise en œuvre envisagée, l'objectif final est de permettre aux étudiants d'apporter une réponse au problème posé, de la matérialiser sous la forme d'un langage informatique, puis de la tester dans un environnement réaliste. Reste à la charge des enseignants le choix de déterminer les modalités d'encadrement (à distance ou en présentiel), la méthode de travail des étudiants (en autonomie, en binômes ou en groupe), etc. Toutefois, dans la mesure

où l'interaction directe entre le joueur et le jeu est autorisée, l'évaluation ne doit pas porter sur le résultat de la compétition (nombre de victoires et de défaites) mais bien sur l'analyse du code réalisé. En effet, il est difficile de déterminer dans quelle proportion une victoire est due au programme réalisé ou au simple talent de joueur d'un étudiant. La compétition finale est donc là comme un élément de motivation et non comme un outil d'évaluation.

Quelque soit le mode de jeu utilisé, la création de situations de jeu est nécessaire pour refléter le déroulement de la campagne ou le contexte d'une compétition. Pour aider à la conception de ces situations de jeu, un squelette de scénarios est proposé.

### Squelette de scénarios

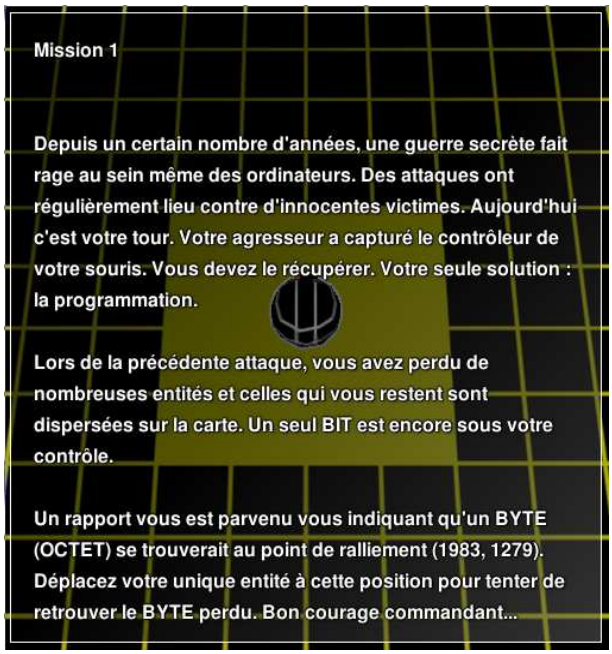



FIGURE 4.21 – Exemple de briefing (Mission 1)

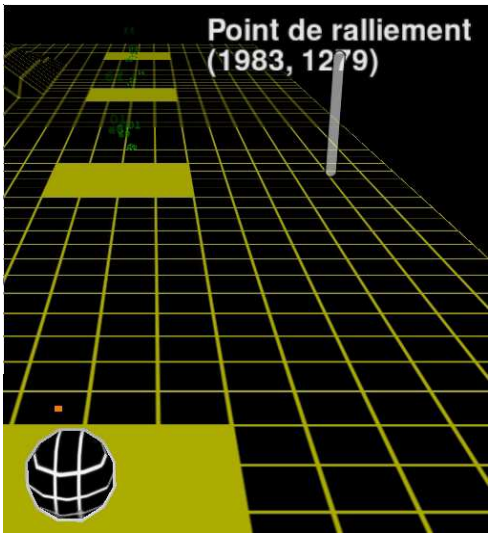



FIGURE 4.22 – Exemple d'affordance dans la mission 1.
*Le marqueur « Point de ralliement (1983, 1279) » aide le joueur à comprendre où déplacer le BIT.*

Pour permettre aux professeurs d'adapter le jeu à leurs enseignements, ils doivent pouvoir créer de nouvelles situations de jeu. À titre d'exemple, réaliser une mission d'une campagne consiste à intégrer le texte de briefing dans le jeu (voir figure 4.21), positionner les unités du joueur ainsi que les unités alliées et ennemies sur la carte, positionner le point de vue, ajouter de l'affordance pour aider à la compréhension de la scène (voir figure 4.22) et définir les conditions de victoire et de défaite. À l'origine, les premières versions de scénarios étaient intégrés

directement dans le moteur du jeu. Cette solution rendait leur maintenabilité et leur évolutivité difficile car elle nécessitait une recompilation systématique du moteur. Pour faciliter la création et la maintenance de nouveaux scénarios de jeu, l'interpréteur Lua de Spring a été utilisé.

En effet, *Spring* dispose d'un interpréteur Lua permettant aux utilisateurs de structurer des scripts au sein d'une archive chargée et exécutée dynamiquement par le moteur. Tous les jeux fonctionnant sur *Spring* sont basés sur cette technologie. Pour permettre aux enseignants de créer des scénarios adaptées à leurs enseignements un squelette d'archive leur est proposé. Le squelette contient quelques scripts prédéfinis qui gèrent l'exécution du scénario et l'affichage de menus. Pour construire un scénario de jeu, l'enseignant doit simplement implémenter les fonctions suivantes : la fonction *Start* sert à initialiser le scénario courant ; la fonction *Show-Briefing* permet de définir le contexte de jeu ; la fonction *Update* est utilisée pour définir des évènements au cours de jeu et déterminer les conditions de victoire et de défaite ; la fonction *Stop* doit nettoyer l'ensemble des données allouées pour le scénario courant. Chacune de ces fonctions sont automatiquement appelées au cours de la simulation. Une campagne est donc une structuration séquentielle de scénarios alors qu'une compétition est un simple scénario conçu pour être fonctionnel en mode multijoueur.

Pour implémenter ces fonctions, les enseignants utilisent les interfaces Lua de *Spring* [50]. Pour des problèmes de sécurité, *Spring* possède trois contextes d'exécution : le « *LuaUI* », le « *LuaRules (unsynced)* » et le « LuaRules (synced) ». En fonction du contexte, certaines interfaces ne sont pas actives. Le code des scénarios étant exécuté dans le contexte « *LuaRules (unsynced)* », seules les interfaces compatibles dans ce contexte sont utilisables [51].

Ce squelette a initialement été réalisé comme base à la conception de campagnes. Toutefois, Il peut être utilisé pour préparer les compétitions du mode escarmouche. Dans tout les cas, l'objectif est de simplifier la conception de campagnes et de compétitions pour permettre aux enseignants de construire de nouvelles ressources pédagogiques adaptées à leurs étudiants.

50. Interfaces Lua de *Spring* : `http://springrts.com/wiki/Lua_Scripting`.
51. Validité des interfaces Lua de Spring en fonction du contexte d'exécution : `http://springrts.com/wiki/Category:Lua`.

## Conclusion

Le jeu sérieux, évalué dans le chapitre suivant, est donc une combinaison des trois entités suivantes : *Spring*, Prog&Play et un mode de jeu (campagne ou escarmouche) (voir figure 4.23).

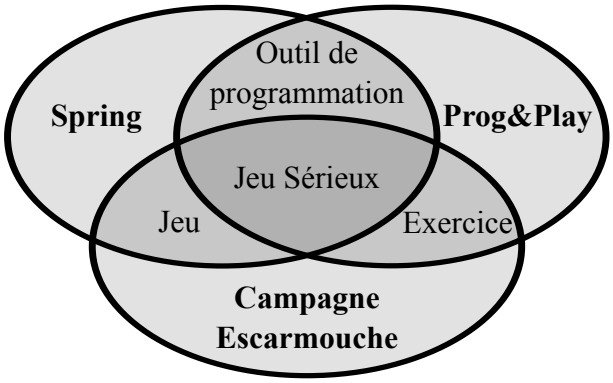



FIGURE 4.23 – Composition du jeu sérieux.

Ce jeu sérieux est donc basé sur la seconde version de Prog&Play. L'intégration de l'API dans les moteurs *Spring* et *WarZone 2100* illustre la compatibilité de cette version avec plusieurs jeux de STR. Pour valider la portabilité totale de l'API, il reste à étudier le portage de la partie « Client » de Prog&Play vers différents langages de programmation. L'interface bas niveau de la partie « Client » est disponible en langage C. Par conséquent, tout langage capable de charger une bibliothèque C peut utiliser le système Prog&Play. Actuellement, des interfaces pour les langages C, C++, Java, Ada, OCaml, Scratch et Compalgo ont été développées.

# 4.5 Portage de la partie « Client » de Prog&Play vers différents langages de programmation

Conformément à la présentation de l'informatique en tant que discipline scientifique, l'architecture des enseignements introductifs (voir section 3.1.2) peut être structurée en six approches différentes : impérative, orientée objet, fonctionnelle, étendue, algorithmique et matérielle. Les langages de programmation utilisés dans ces approches peuvent être impératifs, orientés objet ou fonctionnels. Pour montrer la portabilité de Prog&Play dans les différentes méthodologies d'enseignements, cette section présente l'interfaçage de l'API Prog&Play avec quelques langages de programmation compatibles avec les différentes approches.

### 4.5.1 Prog&Play en C/C++

L'API Prog&Play, telle qu'elle a été présentée en détail dans la section 4.2.2, est disponible à travers la librairie « pp-client » dont l'utilisation est possible grâce aux interfaces « PP_Client.h » et « PP_Error.h » (voir figure 4.24, le détail de ces interfaces est disponible en annexe A). Dans une approche impérative, il est possible d'utiliser directement cette interface pour interagir avec le jeu vidéo.

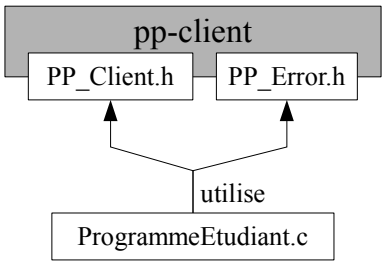



FIGURE 4.24 – Graphe de dépendance de Prog&Play pour programmer en C.

Les liens étroits entre le C et le C++ rendent d'autre part l'utilisation de l'API possible dans un contexte orienté objet (OO). Le diagramme de classe de la figure 4.25 présente une proposition de l'API sous une forme OO. La classe « PP » (Prog&Play) fournit les méthodes nécessaires pour accéder à l'ensemble des données du jeu comme la position de départ, le niveau des ressources disponibles ou l'ensemble des unités propres à une coalition.

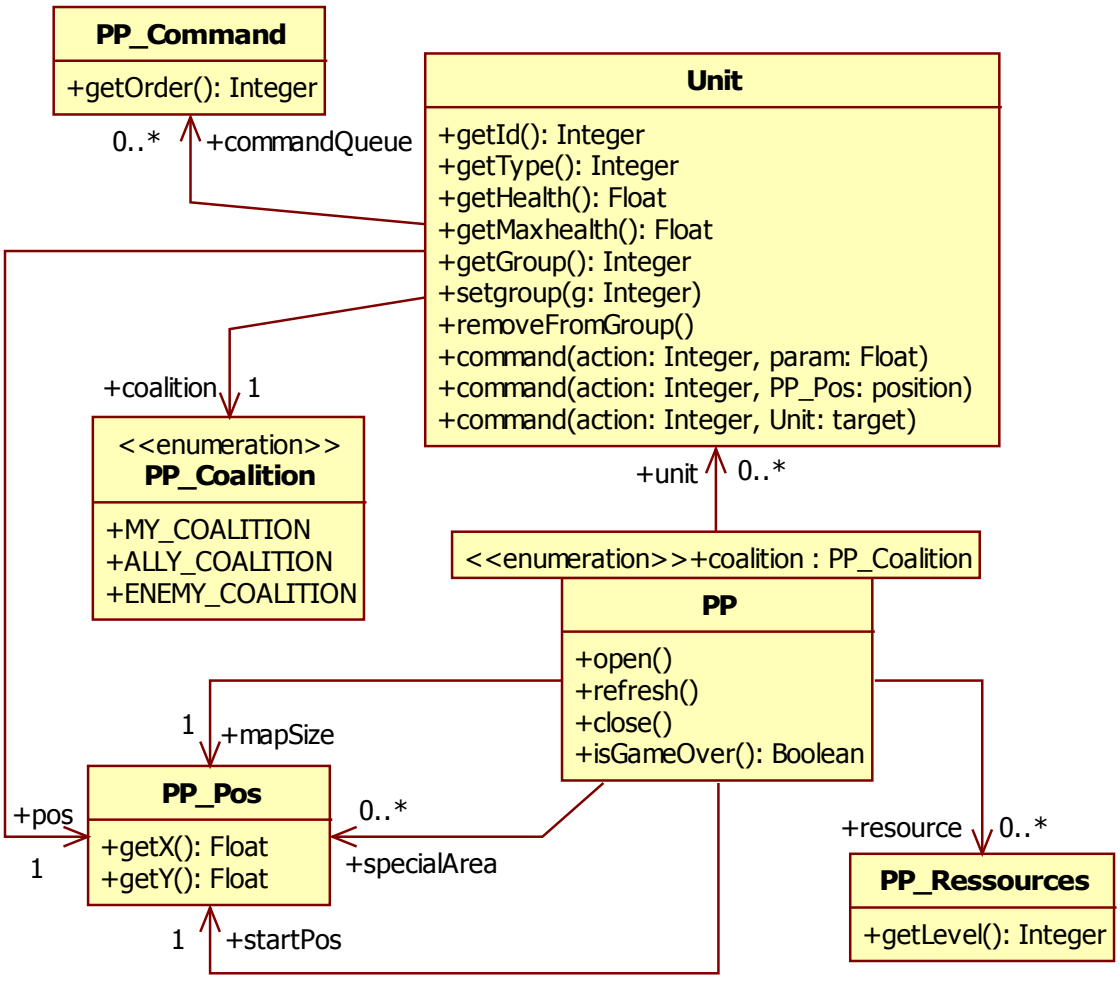



FIGURE 4.25 – Proposition d'une représentation de l'API Prog&Play sous une forme orientée objet.

### 4.5.2 Prog&Play en Java

Le langage de programmation Java est, à l'image du C, un langage utilisé dans de nombreuses formations pour aborder les concepts orientés objets. Le langage Java fournit un mécanisme (le JNI - *Java Native Interface*) pour appeler depuis la JVM (*Java Virtual Machine*) des bibliothèques réalisées en langage C. La figure 4.26 présente le graphe de dépendance entre les différentes interfaces. La librairie « pp_java » grâce à JNI adapte les fonctions et types manipulés par la librairie native C (« pp-client ») en fonctions et types de données exploitables en Java. Les fichiers « Unit.java » et « PP.java » implémentent respectivement les classes « Unit » et « PP » proposées dans la figure 4.25 et accèdent aux données du jeu via la classe implémentée dans le fichier « PP_Native.java ». À titre d'exemple, une position est représentée en C par une structure composée de deux réels x et y. Le transfert de cette structure en Java s'effectue par l'intermédiaire d'un tableau de réels qui est à son tour recompacté sous la forme d'une instance de classe représentant une position.

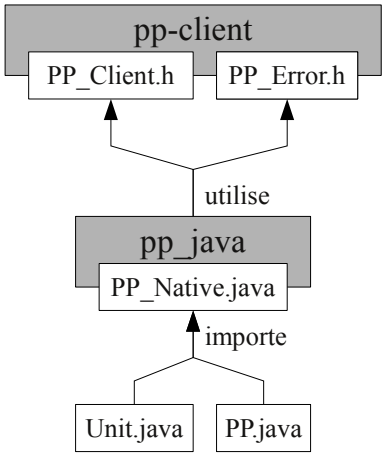



FIGURE 4.26 – Graphe de dépendances de Prog&Play pour programmer en Java.

### 4.5.3 Prog&Play en Compalgo

Compalgo tel qu'il a été défini dans la section 3.2.2 est un environnement de programmation conçu à des fins d'initiation à la programmation pour les étudiants de première année de DUT Informatique. Cet environnement de programmation ainsi que son interprète associé sont réalisés en Java. L'intégration de Prog&Play dans Compalgo a donc pu être produite à

l'aide de la librairie « pp_java » (voir figure 4.27). L'extension de l'interprète a consisté à rendre exploitable en Compalgo les types et fonctions accessibles via la classe « PP_Native ». À titre d'exemple, un objet tel qu'une position ne peut être transféré directement à l'interprète Compalgo et doit être décomposé en valeurs de type de base pour être ensuite reconstruit sous la forme d'un enregistrement. Des modifications supplémentaires ont porté sur la gestion du passage des paramètres (entrée, sortie et mise à jour) et l'ajout de fonctions non incluses dans la bibliothèque du langage mais utiles pour la réalisation d'IA (tirer un nombre aléatoire et effectuer une pause).

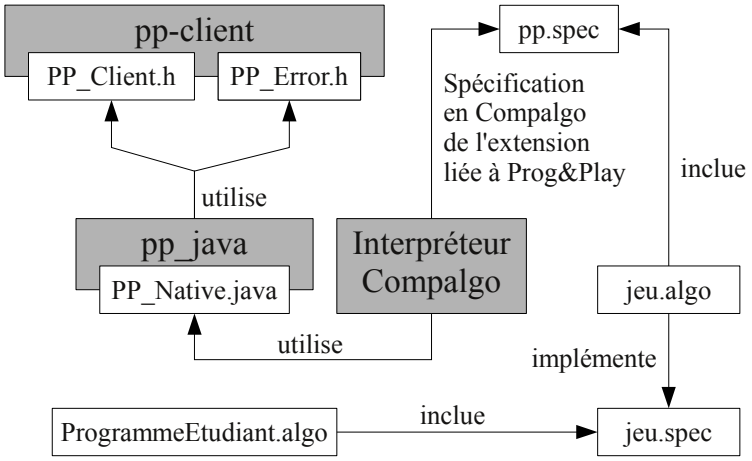



FIGURE 4.27 – Graphe de dépendances de Prog&Play pour programmer en Compalgo.

Le fichier « pp.spec » est la spécification bas niveau en Compalgo de l'extension incluse dans l'interprète. Dans cette spécification, les données sont décomposées en valeurs de type de base. Une interface de plus haut niveau (« jeu.spec ») est donc proposée aux étudiants et présente les données de manière plus naturelle.

### 4.5.4 Prog&Play en OCaml

OCaml (*Objective Caml*) est un langage fonctionnel qui offre des possibilités de programmation impérative, OO et modulaire. Ce langage est caractérisé par un typage statique fort et inféré c'est-à-dire que le type d'une fonction est déterminé automatiquement suivant l'utilisation de ces paramètres. Le filtrage par motif est une autre spécificité qui permet aux programmeurs de manipuler aisément des structures de données complexes. Enfin, en tant que langage fonctionnel, il gère la récursion terminale et permet l'utilisation de fonctions d'ordre supérieur

et de fermetures. La récursion terminale est un cas particulier de la récursivité qui permet de ne pas stocker la pile d'exécution et donc d'économiser de l'espace mémoire. Les fonctions d'ordre supérieur sont des fonctions qui acceptent en entrée une ou plusieurs fonctions et/ou qui en renvoient une. Les fermetures sont des fonctions déclarées à l'intérieur d'une autre fonction et qui font référence aux variables locales ou aux paramètres de la fonction dans laquelle elles sont définies. Le langage OCaml est donc un langage utilisable dans des approches fonctionnelles de l'apprentissage de la programmation.

Le langage OCaml est distribué avec une suite de logiciels dont un interprète interactif, un compilateur, un débogueur et une bibliothèque standard, entre autres. Il est également fourni avec une interface de compatibilité qui permet de lier du code OCaml à des primitives en C. Cette fonctionnalité a été utilisée pour connecter la librairie « pp-client » au module PP représenté par la librairie « pp_ocaml »et le fichier d'interface « pp.ml » (voir figure 4.28). Ce module s'attache à gérer les erreurs et à rendre les données et fonctions manipulables dans un contexte fonctionnel. À titre d'exemple, plutôt que d'être accessibles une à une, les unités d'une coalition sont fournies sous la forme d'une liste exploitable de manière récursive. D'autre part, les fonctions de gestion de l'API telles que les fonctions C `int PP_Open(void)` et `int PP_Refresh(void)` (voir détail en annexe A.2) ont une utilisation purement séquentielle. Elles sont donc inexistantes dans l'interface OCaml et sont gérées en interne dans la bibliothèque « pp_ocaml ». Ainsi, la programmation du jeu peut être réalisée d'une manière purement fonctionnelle.

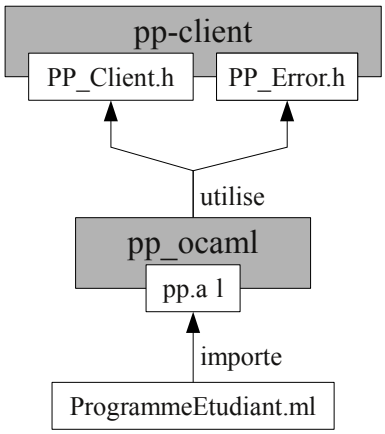



FIGURE 4.28 – Graphe de dépendances de Prog&Play pour programmer en OCaml.

### 4.5.5 Prog&Play en Ada

Ada est un langage conçu dans le début des années 1980 pour le département de la Défense des États-Unis. En raison d'un typage fort et de la structuration du code autour des paquetages, ce langage est souvent utilisé pour l'enseignement de la programmation car il aide les débutants en informatique à acquérir de solides bases en programmation. Dans un contexte d'initiation à l'informatique, Ada peut être employé dans une approche impérative ou en tant que langage d'illustration dans une approche algorithmique.

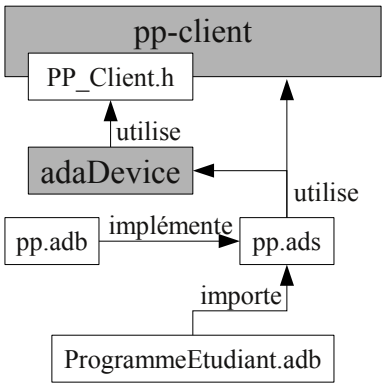



FIGURE 4.29 – Graphe de dépendances de Prog&Play pour programmer en Ada.

Ada permet en outre d'utiliser des composants écrits dans d'autres langages tel que le C grâce au paquetage « Interfaces.C » qui définit des types compatibles. La figure 4.29 illustre la connexion directe qui peut s'établir entre du code Ada (« pp.ads ») et des fonctions C implémentées dans la librairie (« pp-client »). Le module « adaDevice » permet de résoudre une subtilité liée à l'appel de la fonction C « `PP_Unit_ActionOnPosition` ». Cette dernière prend comme troisième paramètre une position définie comme étant une structure composée des champs x et y. Or le passage des structures de Ada à C se fait par adresse, il a donc fallu créer une couche supplémentaire (« adaDevice ») adaptée à ce type de passage qui fait l'interface vers la fonction C ciblée.

### 4.5.6 Prog&Play en Scratch

Scratch est écrit à l'aide de Squeak, une implémentation libre du langage Smalltalk-80. Squeak se présente comme un environnement de programmation dynamique et fournit la possibilité de charger des primitives externes écrites en C. Le module « ProgAndPlayPlugin » (voir

figure 4.30) redéfinit l'ensemble des fonctions de la librairie « pp-client » pour qu'elles soient exploitables par Squeak.

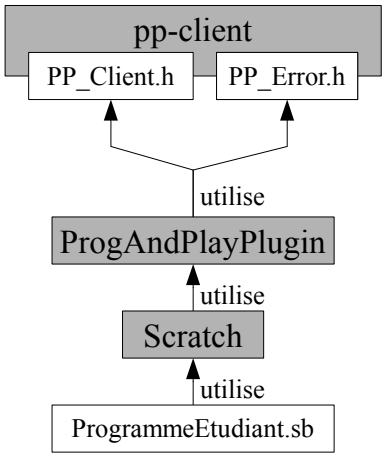



FIGURE 4.30 – Graphe de dépendances de Prog&Play pour programmer en Scratch.

Outre cette redéfinition, l'interface graphique de Scratch a été modifiée pour y intégrer une nouvelle catégorie de blocs nommée « Progandplay ». Cette dernière, identifiable par sa couleur rouge, donne accès aux briques dédiées à la manipulation de l'API Prog&Play (voir figure 4.31). Dans Scratch, les blocs dits « rapporteurs » ne peuvent retourner que des types de base tels qu'un nombre, une chaîne de caractères ou un booléen. Par conséquent, et à titre d'exemple, la fonction C qui retourne la position de départ du joueur sur la carte est accessible dans Scratch à travers deux briques distinctes. Elles rapportent respectivement les valeurs $x$ et $y$ de la position de départ :

En-tête de la fonction C : `PP_Pos PP_GetStartPosition(void);`
Briques Scratch correspondantes : position x de départ et position y de départ.

Scratch tel qu'il a été défini dans la section 3.2.2 est un environnement de programmation utilisable par des enfants pour créer leurs propres histoires animées, jeux vidéo ou créations interactives. Toutes les nouvelles briques introduites avec la catégorie « Progandplay » sont bien sûr combinables avec les autres blocs du langage de façon à respecter les applications initiales de Scratch.

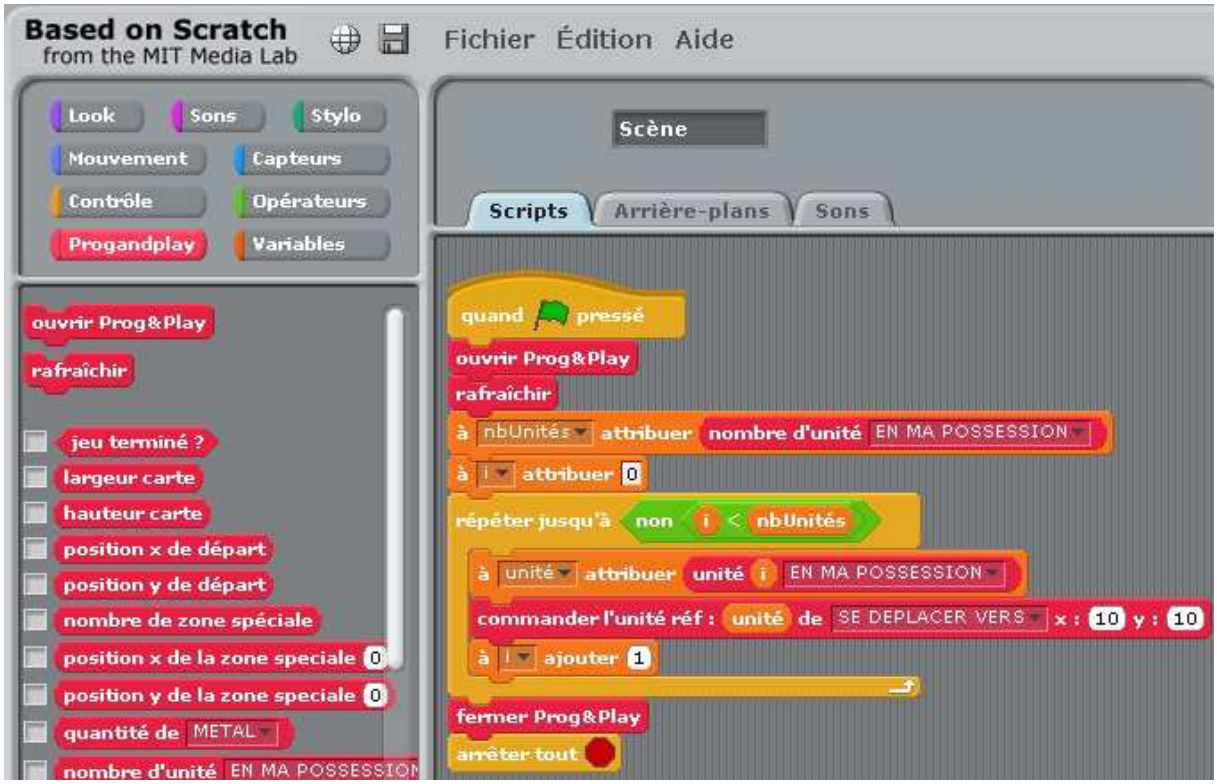



FIGURE 4.31 – Interface de Scratch modifiée pour utiliser Prog&Play.
*Le programme en Scratch présenté dans cette image déplace l'ensemble des unités du joueur à la position (10, 10).*

# Conclusion

La portage de la partie « Client » de Prog&Play vers différents paradigmes de programmation ouvre le jeu à l'ensemble des approches pédagogiques de l'enseignement des fondamentaux de la programmation. L'API, telle qu'elle a été présentée dans la section 4.2.2, est donc compatible avec des langages impératifs, fonctionnels et OO qu'ils soient interprétés ou compilés.

Ces interfaces sont des illustrations du champ des possibles. Ces langages sont issus pour la plupart des formations dans lesquelles des expérimentations ont été menées, en particulier pour le C, Java, Compalgo et OCaml. Les interfaces développées en Ada et Scratch semblent être particulièrement intéressantes pour l'apprentissage de la programmation. En particulier l'interfaçage de Prog&Play avec Scratch ouvre des pistes de recherche intéressantes sur l'évaluation de la combinaison de ces deux technologies. Selon la démarche exposée, il est possible soit

de concevoir d'autres interfaces adaptées à de nouveaux langages de programmation, soit plus simplement d'adapter les interfaces existantes à des contextes d'enseignements particuliers.

L'intégration de la partie « Fournisseur » de Prog&Play dans différents jeux de STR permet d'augmenter la diversité des jeux sérieux réalisables. Pour être utilisable, chaque jeu doit fournir un ensemble de constantes pour identifier chaque type d'unité et chaque action réalisable dans le jeu. Ces spécifications doivent alors être implémentées dans les langages de programmation des différents portages de l'interface « Client ». De cette manière, chaque jeu maintiendra son interopérabilité avec l'ensemble des langages compatibles avec Prog&Play.

La conception d'un jeu sérieux sur les fondamentaux de la programmation a donc été réalisé grâce à *Kernel Panic* et à l'intégration de Prog&Play dans *Spring*. Les différentes interfaces de l'API, disponibles dans différents langage, permettent alors d'utiliser le jeu dans l'ensemble des approches pédagogiques et aide à la recherche de contextes d'expérimentation.

# 5

# Évaluation

Le jeu étant opérationnel, il reste à évaluer son efficacité et sa pertinence en contexte d'enseignement. Ce processus étant une tâche complexe, il convient de construire un modèle d'évaluation afin de pouvoir mener différentes expérimentations dans divers contextes. Chaque évaluation participe à l'amélioration de l'outil.

## Introduction

L'ingénierie didactique de Cobb *et al.* [CCD+03] sert de support à l'évaluation du jeu sérieux. Cette méthodologie vise à expérimenter de nouvelles formes d'apprentissages à travers la mise en œuvre de moyens spécifiques. Dans cette approche, le chercheur tente de valider une théorie « humble » concernant les processus d'apprentissage et envisage d'effectuer une série d'expérimentations pour la tester. Pour chaque itération, le principe consiste à construire, en collaboration avec les protagonistes, une ingénierie didactique, à la mettre en œuvre au cours d'une ou plusieurs expérimentations en procédant à de multiples observations puis à analyser les résultats afin de proposer une éventuelle nouvelle ingénierie didactique destinée à tester de nouveaux éléments concernant la théorie. Chaque itération permet de faire varier différents paramètres supportés par la théorie. L'objectif n'est pas de construire une ingénierie didactique idéale mais bien de valider une théorie concernant l'apprentissage.

Dans le cas présent, il s'agit de valider l'hypothèse selon laquelle l'apprentissage de la programmation au travers d'un jeu sérieux contribue à motiver les étudiants et les encourage à poursuivre leur formation en informatique. L'idée à transmettre serait que l'activité de program-

mation peut être agréable en soi et que son existence n'est pas circonscrite au cadre strictement scolaire de la réalisation d'exercices.

L'application de cette méthodologie à l'évaluation du jeu sérieux conduit ainsi à envisager plusieurs itérations. Chaque itération correspond à un contexte différent d'expérimentation de l'outil associé à une ingénierie didactique spécifique mettant en œuvre une exploration particulière de la théorie à valider. De plus, à chaque itération, un ensemble de critères d'évaluation doit être envisagé pour mesurer les effets de l'outil en regard des objectifs fixés.

## 5.1 Première expérimentation

Cette première expérimentation a pour objectif d'observer le comportement des étudiants vis-à-vis du jeu sérieux afin de déterminer s'il possède réellement un potentiel motivationnel. Pour réaliser la première itération nous avons choisi un environnement connu nous permettant d'apprécier l'impact de l'utilisation du jeu sérieux sous couvert de notre propre pratique enseignante. Ainsi cette première expérimentation a été réalisée dans le cadre d'un cours d'algorithmique ordinaire délivré dans un contexte familier.

### 5.1.1 Contexte de l'expérimentation

L'expérience a eu lieu avec des étudiants de première année en informatique de l'IUT « A » de Toulouse. Sur une promotion de cent quatre-vingt seize étudiants, quinze étudiants parmi quarante volontaires ont été sélectionnés. La sélection s'est effectuée à partir d'un questionnaire destiné à évaluer leur motivation pour jouer aux jeux vidéo et pour apprendre la programmation. Ce questionnaire est basé sur les buts de Viau [Via97] ; le modèle de valeur et de succès des aspirations de Bandura [Ban97] ; et le modèle causal de Pintrich et Schunk [PS96]. Chaque modèle suggère des indicateurs spécifiques que nous avons adaptés à la motivation des étudiants pour pratiquer la programmation et les jeux vidéo. Nous nous sommes inspirés du questionnaire « *motivated strategies for learning* » [PMB93].

Ces étudiants sont novices en programmation. Au début de l'expérimentation, ils connaissent un seul langage de programmation : le Compalgo. Durant cinq séances d'une heure trente minutes, les étudiants ont ainsi utilisé l'API Prog&Play avec le langage Compalgo dans un environnement Windows. Lors de la réalisation de cette expérimentation, la campagne du jeu sérieux n'était composée que de cinq missions à savoir les missions 1, 3, 4, 6 et 7 présentées

page 106. Les briefings étaient alors légèrement différents de manière à présenter un scénario cohérent.

Conformément au programme pédagogique national du DUT Informatique [Min05], les enseignements du tronc commun de la « formation initiale en quatre semestres » sont structurés en deux groupes. Le premier est constitué par le champ disciplinaire « Informatique » (noté Info). Le second apporte les « Connaissances et Compétences Générales » (noté CCG). Lors des deux premiers semestres, ces deux groupes sont composés de trois unités d'enseignements chacun. L'« Informatique » regroupe l'AP (Algorithmique et Programmation), l'ASR (Architecture, Systèmes et Réseaux) et l'OMGL (Outils et Modèles du Génie Logiciel). le CCG regroupe : les mathématiques ; l'économie et gestion des organisation ; les langues, l'expression et la communication. L'expérimentation a été conduite dans le cadre des enseignements d'AP au début du second semestre.

La définition d'une ingénierie didactique et de critères d'évaluation constitue un préalable pour évaluer cette première confrontation du jeu sérieux avec des étudiants en contexte réel.

### 5.1.2 Conception du protocole d'évaluation

L'ingénierie didactique, conçue pour cette première itération, a été divisée en deux phases (voir tableau 5.1) : la première consiste à **laisser les étudiants jouer** à *Kernel Panic* sans utiliser la programmation afin qu'ils puissent découvrir l'univers du jeu, les unités et leurs caractéristiques. La deuxième phase concerne l'activité de programmation proprement dite : après une présentation de l'API Prog&Play, les étudiants s'attachent à résoudre **les cinq missions en réalisant de petits programmes**. Pour chaque mission, l'enseignant présente les objectifs de jeu et aide les étudiants à concevoir une solution algorithmique du problème. Après avoir conçu un programme résolvant la mission courante, l'étape d'institutionnalisation permet de récapituler l'ensemble des concepts abordés lors de la mission. Après ces deux premières phases, les étudiants sont théoriquement capables de réaliser de courts programmes compatibles avec le jeu *Kernel Panic*.

Eu égard aux objectifs de cette expérimentation, trois critères d'évaluation ont été définis : l'**apprentissage de la programmation** ; l'**amusement** ; et l'**utilisabilité du système**. Les deux premiers critères évaluent le concept de « jeu sérieux » sous l'angle de l'apprentissage de la programmation et du divertissement procuré par le jeu. Le dernier critère permet d'identifier les composants du jeu qui peuvent perturber les étudiants.

TABLE 5.1 – Organisation des enseignements.

<table>
<tr><th>Phase 1</th><th>Phase 2</th></tr>
<tr><td>Présentation du jeu initial et jeu en réseau</td><td>Présentation de l'API Prog&Play et réalisation des cinq missions de la campagne<br><table><tr><th>Pour chaque mission</th></tr><tr><td>(1) Présentation des objectifs<br>(2) Les étudiants programment leur(s) solution(s)<br>(3) Institutionnalisation</td></tr></table></td></tr>
</table>

Les paramètres de l'évaluation ont été fixés a priori, en définissant les critères et les indicateurs de l'évaluation avant l'expérimentation (approche « *ex ante* »). Leur organisation temporelle lors de l'expérimentation est présentée dans la figure 5.1. Par souci de lisibilité, le détail des indicateurs est présenté au fur et à mesure de leur analyse lors de cette première itération.

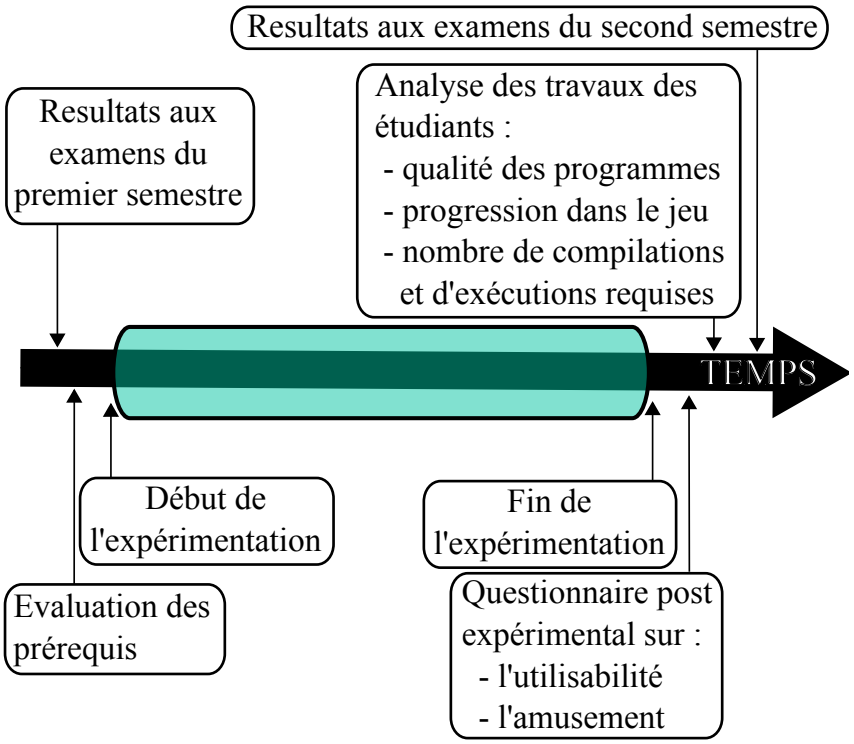



FIGURE 5.1 – Organisation temporelle de l'évaluation pour la première expérimentation.

### 5.1.3 Résultats

#### Évaluation des prérequis

Le premier indicateur présenté concerne l'évaluation des prérequis. Il n'est pas lié directement à l'un des trois critères d'évaluation, mais a pour but de déterminer le profil des étudiants. Cette évaluation a été construite à partir des travaux de Ginat *et al.* [Gin04] sur la boucle algorithmique. Les étudiants répondent à huit questions progressives, sur papier en une vingtaine de minutes et sans documentation. Ce test est proposé après la phase de jeu et son évaluation n'est pas prise en compte pour la validation du semestre.

Les solutions des étudiants ont été analysées d'après les critères de Smith et Cordova [SC05] en portant une attention particulière sur les solutions produites et les sorties attendues. Les identifiants doivent être explicites et suivre les conventions de nommage et les programmes construits doivent être simples et élégants. Enfin, une attention est portée à l'indentation, aux commentaires et à la lisibilité du code.

Afin d'évaluer la pertinence de ce test, les résultats des étudiants à cet exercice sont comparés à leur résultat de l'examen d'« algorithmique et programmation » du premier semestre. Cette comparaison est visualisée dans la figure 5.2 et met en évidence pour chacun des quinze étudiants, leurs notes (entre 0 et 20) à l'examen d'algorithmique et à cette évaluation des prérequis. A l'exception des étudiants 6 et 14, l'évaluation semble refléter leur niveau de réussite (l'erreur moyenne est de 2,06 points).

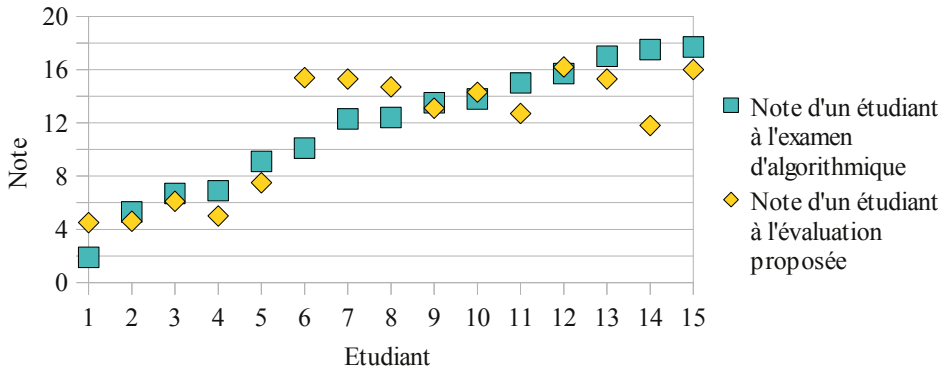



FIGURE 5.2 – Comparaison des résultats des étudiants à notre évaluation et à l'examen d'algorithmique.

**Critère 1 : Apprentissage de la programmation**

L'évaluation de l'apprentissage est basée sur les travaux de Chen et Cheng [CC07], Gestwicki et Sun [GS08] et Leutenegger et Edgington [LE07]. Ces travaux ont permis de définir trois indicateurs : l'acquisition de connaissances, la qualité des programmes et la quantité de travail fourni.

**Acquisition de connaissances :** pour évaluer l'impact du jeu sur l'apprentissage des étudiants, la variation des résultats des étudiants entre les examens du premier et du second semestre est comparée entre ceux qui ont utilisé le jeu sérieux et ceux qui ne l'ont pas utilisé. Ces examens sont conçus par des enseignants externes au projet.

En raison du faible échantillon d'étudiants dans cette première expérimentation, une comparaison statistique des deux populations est non valable. Toutefois, il est apparu que les étudiants ayant participé à cet atelier dont le niveau en algorithmique était faible, ont plutôt amélioré leurs résultats au second semestre (voir tableau 5.2).

TABLE 5.2 – Évolution moyenne des notes entre les examens du premier et du second semestre pour les étudiants qui ont participé à l'atelier et dont le niveau algorithmique était faible au premier semestre (note d'AP au S1 < 10).

| | **AP** | **Info** | **CCG** |
|---|---|---|---|
| **Évolution moyenne** | + 1,77 | + 1,62 | + 0,24 |

**Qualité des programmes :** la qualité des programmes est évaluée d'après les critères de Smith et Cordova [SC05]. Parmi l'ensemble des critères définis par ces auteurs, les critères suivants sont retenus pour cette première itération centrée sur les fondamentaux de la programmation : l'exactitude des programmes, le style de programmation et la documentation des programmes et de leur conception.

Pour cette expérimentation, l'indicateur n'a pas fourni de données pertinentes. En effet, le niveau débutant des étudiants évalués, les problèmes relativement fermés des missions proposées et le mode d'encadrement des séances ont contribué à homogénéiser les solutions récupérées.

**Quantité de travail fourni :** la quantité de travail réalisé est évaluée en fonction de la progression des étudiants dans le jeu (nombre de missions terminées) et pour chaque mission, du nom-

bre de compilations et d'exécutions nécessaires à l'obtention d'une solution fonctionnelle. Ces données ont pu être récupérées après avoir intégré un « espion » à l'environnement de développement utilisé lors des séances d'enseignement. Grâce à ces données, nous pouvons définir, pour chaque mission « $m$ », un indice de difficulté (noté « $d_m$ ») présenté dans la figure 5.3. Cet indice de difficulté est calculé comme indiqué dans l'équation 5.1, où « $nbEtudiants_m$ » est le nombre d'étudiants ayant terminé la mission « $m$ » et « $nbCompilations_{im}$ » et « $nbExecutions_{im}$ » sont respectivement le nombre de compilations et d'exécutions requises par le i$^{\text{ème}}$ étudiant pour compléter la mission « $m$ ».

$$d_m = \frac{\sum_{i=1}^{nbEtudiants_m} nbCompilations_{im} + nbExecutions_{im}}{nbEtudiants_m} \tag{5.1}$$

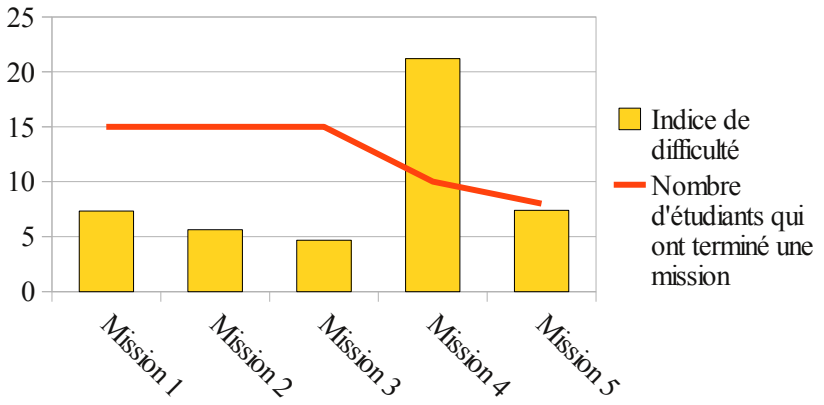



FIGURE 5.3 – Indice de difficulté pour chaque mission.

Contrairement aux attentes, la difficulté des missions n'est pas régulière. Un écart important existe entre la troisième et la quatrième mission. Par conséquent, quelques étudiants se sont retrouvés bloqués et n'ont pu terminer la campagne. Du point de vue du jeu, les dernières missions doivent présenter un niveau de difficulté supérieur aux précédentes de manière à proposer un défi au joueur. Mais cette difficulté doit être progressive.

La baisse de l'indice de difficulté entre les missions 1 et 3 montre l'appropriation progressive de l'API par les étudiants et leur capacité à la combiner avec des structures de contrôle simples.

**Critère 2 : Amusement**

Ce critère a été évalué au travers d'un questionnaire soumis aux étudiants à l'issue de l'expérience. Les questions sont présentées figure 5.4.

– Q1 : Avez-vous apprécié le scénario de la campagne (les missions) ? (1 = « Pas du tout », 7 = « Beaucoup »)
– Q2 : Pensez-vous que l'utilisation de la programmation dans *Kernel Panic* augmente le divertissement ?
  ☐ Réduit le divertissement ☐ Ne change rien
  ☐ Augmente le divertissement ☐ Je ne sais pas

FIGURE 5.4 – Extrait du questionnaire post expérimental en rapport avec l'amusement.

La réponse moyenne pour la première question (Q1, figure 5.4) est de 5,85 et montre que les étudiants ont apprécié la campagne. D'une manière informelle, nous pensons que l'intérêt porté aux missions est en partie dû à la première phase de l'expérimentation. En effet, la session de jeu en réseau a permis aux étudiants de s'approprier l'univers du jeu et a rendu l'immersion des étudiants dans le scénario plus facile.

La seconde question (Q2, figure 5.4) est cruciale. Introduire des concepts algorithmiques dans le jeu ne doit pas en réduire le divertissement. Cette condition est due à la définition des jeux sérieux qui doivent être avant tout divertissants. Or, 100% des étudiants ont trouvé que l'utilisation de la programmation dans le jeu augmentait le divertissement.

**Critère 3 : Utilisabilité du système**

L'intégration de l'API Prog&Play dans *Spring* est fonctionnelle car aucun bogue critique n'a été révélé durant l'expérience. Reste à étudier l'impact du jeu sur le degré de satisfaction des étudiants après l'avoir utilisé. Pour ce faire, la hiérarchie des besoins du joueur de Siang et Rao [SR03] présentée dans la section 1.2.2 a été analysée dans le contexte du jeu sérieux. La figure 5.5 montre la satisfaction des étudiants pour chaque niveau de la pyramide pour notre jeu sérieux.

La plus faible satisfaction concerne le besoin d'esthétique (6) avec une satisfaction moyenne de 47%. Le graphisme vectoriel de *Kernel Panic* semble être moyennement apprécié par les étudiants. Néanmoins, ce STR simplifié permet un apprentissage rapide et constitue un avantage

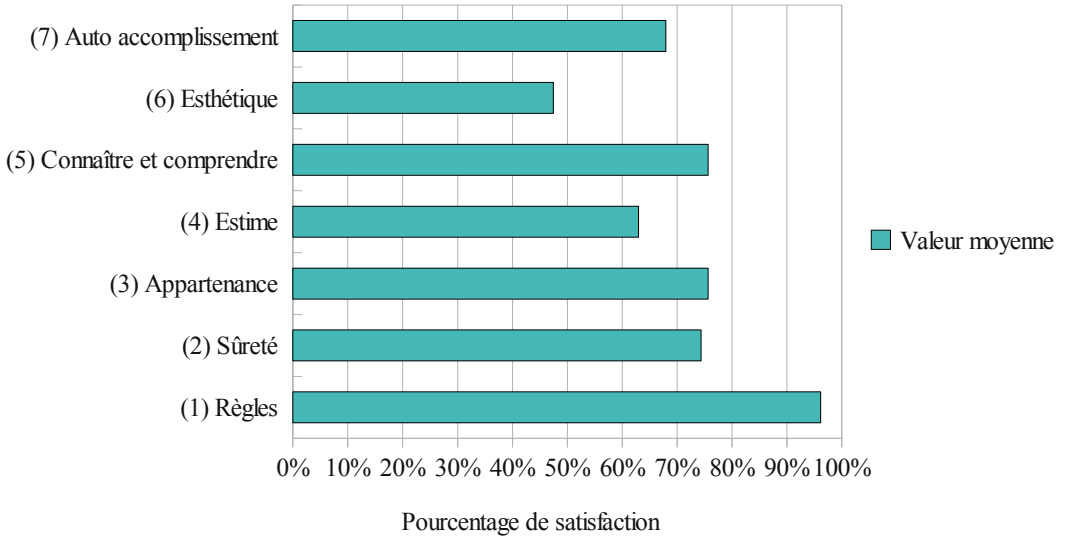



FIGURE 5.5 – Satisfaction moyenne des étudiants pour chaque niveau de la pyramide des besoins (IUT A département informatique Toulouse III).

pour les bas niveaux de la pyramide. Le choix de *Kernel Panic* comme base au jeu sérieux est donc validé.

Actuellement, les informations d'aide à propos de la programmation, le contenu d'apprentissage et l'utilisabilité du jeu ne sont pas incluses dans le jeu et doivent être ajoutées par l'enseignant. Le second niveau (besoin de sûreté) est donc résolu, en grande partie, par l'encadrant.

### Synthèse de la première expérimentation

Cette expérimentation a permis de vérifier que le jeu est amusant et motivant : amusant car tous les étudiants ont trouvé que l'utilisation de la programmation dans le jeu augmente le divertissement ; et motivant car même si une session de jeu en réseau était initialement prévue lors de la dernière séance, plus de la moitié des étudiants ont préféré continuer à programmer plutôt que de jouer. En conclusion, cette première itération a permis de valider le jeu sérieux tant sur son utilisabilité que sur son potentiel ludique.

## 5.2 Deuxième expérimentation

Cette deuxième expérimentation a pour objectif de mettre en application le jeu dans un contexte d'enseignement différent de manière à vérifier sa portabilité. Une attention particulière est portée à l'observation de l'enseignant de manière à déterminer l'impact de l'introduction du jeu dans ses activités.

### 5.2.1 Contexte de l'expérimentation

Cette expérimentation a été réalisée dans le cadre d'un soutien pour les étudiants de l'université Paul Sabatier (Toulouse III) inscrits en première année de licence STS (Science, Technologies, Santé) qui suivent au second semestre la majeure IMM (Informatique, Mathématiques, Mécanique). Ce soutien était organisé de la manière suivante : toutes les deux séances de travaux dirigés (TD), une évaluation (sous la forme de QCM) est distribuée aux étudiants pour jauger la progression de leur apprentissage par rapport à l'évaluation précédente. Les étudiants qui n'atteignent pas la moyenne, sont invités à participer au soutien. Les évaluations proposées sont gérées par l'équipe pédagogique de la formation et n'ont aucun lien avec le jeu sérieux.

Le soutien n'étant pas obligatoire, peu d'étudiants font la démarche de venir aux séances. Dans ce cadre, deux types de soutien leur ont été proposés : un premier classique et un second avec le jeu sérieux.

Pour respecter le contexte d'enseignement utilisé au second semestre de cette formation, les étudiants manipulent le jeu dans un environnement Linux (Mandriva 2008) à l'aide de l'interface en langage C et de l'éditeur de texte « Kate ». À l'image de la première expérimentation, la campagne n'est composé que de cinq missions à savoir les missions 1, 3, 4, 6 et 7 présentées page 106.

### 5.2.2 Intégration du jeu sérieux dans la formation

Dans cette expérimentation, le jeu est intégré à la formation et doit donc s'adapter au calendrier des enseignements présentés dans le tableau 5.3. Le premier TP a pour objectif de familiariser les étudiants avec le contenu d'un ordinateur en leur permettant de démonter entièrement une unité centrale (processeur, carte mère, disque dur, mémoire vive, etc.).

Le deuxième TP se concentre sur le système Linux avec la présentation des commandes basiques du *shell*. La première séance de soutien prend effet à cet instant alors que les étu-

TABLE 5.3 – Calendrier des enseignements du L1S2 de l'Université Paul Sabatier.

| Semaine | TD | QCM | TP | Soutien |
|---|---|---|---|---|
| ... | | | | |
| 3 | TD1 (Langage algorithmique) | | | |
| 4 | TD2 (Tableaux de situations) | QCM1 | TP1 (Architecture) | |
| 5 | TD3 (Introduction au C) | | TP2 (Environnement multi-fenêtre et *shell* Linux) | S1 |
| 6 | TD4 (Raffinage et sous-programmes) | QCM2 | TP3 (Environnement d'exécution, compilation et structures de contrôle) | S2 |
| 7 | TD5 (Sous-programmes et tableaux) | | TP4 (Tableaux de situation et débogage) | S3 |
| 8 | TD6 (Sous-programmes et types simples) | QCM3 | TP5 (sous-programmes, tableaux et débogage) | S4 |
| 9 | TD7 (Sous-programmes, types complexes et raffinage) | | TP6 (Projet) | S5 |
| 10 | TD8 (Développement d'un cas complexe et raffinage) | QCM4 | TP7 (Projet) | S6 |
| 11 | TD9 (Recherches et tri basique) | | TP8 (Projet) | S7 |
| 12 | | | TP9 (Projet) | S8 |

diants n'ont pas encore eu de travaux pratiques de programmation. Par conséquent, cette première séance se consacre à la présentation de l'univers du jeu à travers la réalisation d'une mini compétition. Cette phase correspond à l'étape 1 de l'organisation des enseignements (voir tableau 5.1 page 126) de la première expérimentation. À cette occasion les étudiants manipulent l'environnement Linux et réinvestissent les compétences acquises lors du deuxième TP.

Le troisième TP introduit l'environnement d'exécution et de compilation ainsi que la syntaxe en langage C des structures de contrôle. Les étudiants ont donc simplement eu une introduction au langage C avant la deuxième séance de soutien. D'autre part, s'ils participent au soutien, ils ont théoriquement eu quelques difficultés lors des premières séances de travaux dirigés. Par conséquent, cette deuxième séance de soutien propose de résoudre les premières missions de la campagne à l'aide du langage algorithmique vu en TD. Une surcouche à l'interface C de l'API Prog&Play a donc été créée (« PP_ALGO.h » - voir annexe D). Elle définit les mots clés du langage algorithmique ainsi que les opérateurs utilisables pour interagir avec le

jeu *Kernel Panic*. Les mots clés de ce langage algorithmique sont les suivants : Debut, Fin, Si, Alors, Sinon, FinSi, TantQue, Faire, FinTantQue, Et, Ou, Non, VRAI et FAUX. Les opérateurs disponibles sont présentés dans le tableau 5.4.

TABLE 5.4 – Opérateurs disponibles pour interagir avec le jeu *Kernel Panic* en langage algorithmique.

| Opérateur | Description |
|---|---|
| OUVRIR_JEU | Ouvre la connexion avec le jeu. Cet opérateur doit être appelé avant tout autre opérateur. |
| FERMER_JEU | Ferme la connexion avec le jeu. Cet opérateur doit impérativement être le dernier opérateur à être appelé avant la fin du programme. |
| current_unit | Représente l'unité en cours d'utilisation. |
| PREMIERE_UNITE | initialise l'unité courante à la première unité contrôlée par le joueur. |
| UNITE_SUIVANTE | Fait passer l'unité courante à l'unité suivante contrôlée par le joueur. |
| DERNIERE_UNITE | Fournit la dernière unité contrôlable par le joueur. |
| EST_UN_BIT | Indique VRAI si l'unité courante est un BIT. |
| EST_UN_BYTE | Indique VRAI si l'unité courante est un BYTE. |
| EST_UN_ASSEMBLEUR | Indique VRAI si l'unité courante est un ASSEMBLEUR. |
| DEPLACER_VERS(xCible, yCible) | Donne l'ordre à l'unité courante de se déplacer vers les coordonnées indiquées. |

Grâce à ce langage algorithmique, les étudiants peuvent réaliser les trois premières missions sur les cinq que contenait la campagne lors de cette expérimentation. L'utilisation du langage algorithmique étudié en TD, pour cette séance de soutien, permet aux étudiants de formuler une première solution aux problèmes posés dans les missions et d'observer les résultats dans le contexte du jeu.

Les figures 5.6 et 5.7 présentent les solutions respectives en langage algorithmique des missions 1 et 3 de la campagne telle qu'elle était proposée au moment de l'expérimentation. Pour simplifier la compilation des programmes écrit, un *Makefile* est fourni aux étudiants et quelques informations sont données sur l'utilisation de la commande *make*.

Sur les séances suivantes, les étudiants transforment leurs solutions algorithmiques en des programmes écrits en langage C. De cette manière, ils mettent en application les concepts de décomposition de problèmes et de raffinage successif.

```
#include "PP_ALGO.h"

/* Mission 1 :
   deplacer l'unique unite a
   la position (1983, 1279)
   pour tenter de retrouver
   l'OCTET perdu. */
Debut
    OUVRIR_JEU;
    PREMIERE_UNITE;
    DEPLACER_VERS(1983,
        1279);
    FERMER_JEU;
Fin
```

FIGURE 5.6 – Solution de la première mission en langage algorithmique.

```
#include "PP_ALGO.h"

/* Mission 3 :
   deplacer toute votre armee a la rencontre
   de l'ASSEMBLEUR */
Debut
    OUVRIR_JEU;
    PREMIERE_UNITE;
    TantQue current_unit != DERNIERE_UNITE
        Faire
        DEPLACER_VERS(256, 1024);
        UNITE_SUIVANTE;
    FinTantQue
    DEPLACER_VERS(256, 1024);
    FERMER_JEU;
Fin
```

FIGURE 5.7 – Solution de la troisième mission en langage algorithmique.

En raison de facteurs externes à l'expérimentation (mouvement social étudiant avec blocage de bâtiments d'enseignement), la conduite des dernières séances s'est trouvé fortement perturbée. Toutefois, il a tout de même été possible de réaliser les observations souhaitées sur les activités de l'enseignant.

### 5.2.3 Résultats

Pour étudier l'impact du jeu sur les activités de l'enseignant, la troisième séance de soutien a été filmée. Le visionnage de cette vidéo a permis d'identifier trois activités principales dont celles dédiées : au jeu - « utilisabilité du jeu » ; au savoir à enseigner - « contenu d'apprentissages » ; et aux autres tâches. Le temps utilisé par l'enseignant pour chacune de ces activités, est détaillé dans le tableau 5.5.

TABLE 5.5 – Durée des activités pour une séance d'enseignement.

| Activités | Durée |
|---|---|
| Utilisabilité du jeu | 22 min 13 s |
| Contenu d'apprentissages | 1 h 3 min 42 s |
| Autres tâches | 41 min 27 s |

L'activité « utilisabilité du jeu » concerne des explications sur le contrôle du jeu (comment lancer le jeu ? Comment déplacer la caméra ?) et sur l'interaction entre les programmes des étudiants et le jeu (comment accéder ou commander une unité ? Comment vérifier si le programme est correct ?). L'activité « contenu d'apprentissages » concerne les explications sur les obstacles de programmation tels que les variables (type, affectation, enregistrement), les fonctions (appel, passage de paramètres), ou les structures de contrôle (conditionnelles, itératives). L'activité « autres tâches » concerne les tâches administratives (accueil des étudiants, appel), des explications sur l'utilisation du système d'exploitation, etc.

Au regard des activités énoncées précédemment, le cœur de l'enseignement repose sur le contenu de l'apprentissage. Cette répartition est opportune, car, les explications liées au jeu ne doivent pas empiéter de manière excessive sur les activités de l'enseignant. D'autre part, le temps consacré à l'utilisabilité du jeu diminue au fur et à mesure que les étudiants deviennent experts dans la manipulation du jeu. Certaines interventions de l'enseignant sont comptabilisées dans différentes activités ce qui explique pourquoi la somme des durées de chaque activité est supérieure à la durée d'une séance (deux heures).

Toutefois, l'activité « Utilisabilité du jeu » représente une part non négligeable des explications notamment lors des premières séances. Une question se pose alors sur la capacité d'un enseignant novice en jeu de STR à utiliser un tel outil dans ses enseignements. D'autant plus que ces explications contribuent fortement à satisfaire le second niveau de la pyramide des besoins. Une réponse à cette question peut venir directement des étudiants. En effet, comme le montre la dernière enquête réalisée (voir section 4.1.1), les jeux de STR sont appréciés par 36% des joueurs soit 27% de la population totale. Par conséquent, un certain nombre d'étudiants maîtrisera rapidement le jeu. Une solution consiste alors à inciter ces étudiants à aider leurs camarades à mieux contrôler l'environnement de jeu. Cette solution permet en outre d'initier la communication entre les étudiants et de favoriser l'entraide. Crenshaw *et al.* [CCMT08] notent à ce propos que le manque de communication et d'échange entre les étudiants est un des facteurs favorisant le désintéressement des étudiants pour l'informatique.

#### Synthèse de la deuxième expérimentation

Malgré les évènements qui ont perturbés les enseignements, cette itération a permis dans un premier temps de valider la portabilité du jeu sérieux dans un contexte différent de la première expérimentation. En effet, tout en respectant l'organisation des enseignements, il a été possible

d'intégrer le jeu avec cohérence dans le calendrier. La création d'une surcouche algorithmique à l'interface C de l'API montre également la possibilité d'adapter les interfaces préexistantes à un contexte particulier.

Dans un deuxième temps, l'observation des activités de l'enseignant met en évidence que le cœur de l'enseignement repose bien sur les concepts de programmation. Il importe, tout de même, de souligner l'importance non négligeable des explications liées à l'utilisation du jeu et donc la nécessité de l'enseignant à maîtriser les jeux de STR ou à savoir solliciter l'aide des étudiants sur ce point.

Enfin cette expérimentation vérifie le potentiel motivationnel du jeu sérieux qui a su attirer une dizaine d'étudiants contre seulement deux pour le soutien traditionnel.

## 5.3 Troisième expérimentation

Cette troisième expérimentation a pour objectif d'étudier la mise en œuvre du jeu a une plus grande échelle, afin d'évaluer l'influence de ce dernier sur la motivation des étudiants et son appréciation par des enseignants externes au projet.

### 5.3.1 Contexte de l'expérimentation

Pour cette expérimentation, le jeu a été intégré dans le tronc commun du parcours IMP (Informatique, Mathématiques, Physique) au premier semestre de la première année de licence STS de l'université Paul Sabatier. Dans cette formation, l'informatique est abordée par une approche fonctionnelle. Les étudiants mettent en application les notions de ce paradigme de programmation lors de six séances de travaux pratiques en environnement Linux et à l'aide du langage de programmation OCaml.

La promotion, composée de plus de trois cent étudiants, est structurée en trois sections chacune divisée en groupe de TD d'une trentaine d'étudiants. Chaque groupe de TD est alors séparé en deux sous-groupes pour les TP. L'affectation des étudiants dans les groupes de TP se fait de manière aléatoire. Une quinzaine d'enseignants prennent part à la formation.

Pour tenter d'évaluer l'influence du jeu sur l'apprentissage de la programmation, la promotion a été divisée en deux pour la réalisation des TP. En accord avec l'équipe pédagogique, les sous-groupes impairs ont servi de groupes témoins et ont réalisé les TP classiques alors que les

sous-groupes pairs ont utilisé le jeu, d'où la nécessité de modifier les supports de TP pour les sous-groupes concernés.

### 5.3.2 Intégration du jeu sérieux dans la formation

La précondition émise par l'équipe pédagogique pour pouvoir valider l'intégration du jeu dans la formation porte sur la capacité de ce dernier à maintenir les concepts enseignés de l'approche classique. Le tableau 5.6 présente le calendrier et les objectifs des enseignements de TP dans la version classique.

TABLE 5.6 – Calendrier des enseignements de travaux pratiques d'informatique au L1S1 de l'Université Paul Sabatier.

| TP | Objectifs |
|---|---|
| 1 | Présentation de l'environnement de travail : le système d'exploitation Linux (Mandriva 2008), le gestionnaire de fenêtres KDE, la boucle interactive de OCaml et l'éditeur de texte Kate. |
| 2 | Réalisation de fonctions arithmétiques : utilisation du filtrage, des n-uplets, des fermetures, de la récursivité et des fonctions déjà réalisées ; initiation à la dichotomie. |
| 3 | Utilisation de la bibliothèque graphique : présentation du concept de bibliothèque, calculs trigonométriques, optimisation et récursivité. |
| 4 | Manipulation des listes : traitement et construction de listes ; utilisation du filtrage et de la récursivité sur les listes ; présentation du concept de prédicat ; tris. |
| 5 et 6 | Gestion d'une base de données : réutilisation de fonctions des TP précédents ; projection, élimination de doublons, sélection, mise à jour et tri ; manipulation de types de données utilisateurs. |

Les modifications apportées pour intégrer le jeux dans les TP, tout en respectant les concepts à enseigner, se concentrent sur les séances 1, 3 et 4. Après avoir été validés par l'équipe pédagogique, les modifications sont les suivantes :

- **TP 1** : Outre la présentation du système Linux, le premier TP est utilisé pour présenter le jeu et son univers. Il explique d'une manière détaillée la procédure à suivre pour réaliser des parties en mode multijoueur. Les étudiants sont invités à tester le jeu en séances libres avant la troisième séance de TP.
- **TP 3** : La troisième séance de TP est centrée sur l'utilisation d'une bibliothèque. Le support a donc été totalement récrit pour présenter l'API Prog&Play ainsi que la campagne. Afin de respecter le type d'exercice présenté dans le support originel, une nouvelle mis-

sion a été ajoutée aux cinq missions déjà existantes. Dans cette expérimentation, la campagne est donc composée de six missions à savoir les missions 1, 2, 3, 4, 6 et 7 présentées page 106. Lors de cette troisième séance, les étudiants sont invités à réaliser les quatre premières missions de cette nouvelle campagne. Par cet intermédiaire, les étudiants s'initient à l'utilisation d'une bibliothèque avec la première mission, réalisent un exercice de trigonométrie dans la deuxième, manipulent le filtrage dans la troisième et la récursivité dans la quatrième.

– **TP 4** : Enfin la quatrième séance de TP a également été complètement récrite. Lors de cette séance les étudiants travaillent à la réalisation de la cinquième mission (soit la mission 6 de la campagne présentée page 107). Ils commencent par créer des fonctions pour manipuler des listes d'unités et les utilisent pour terminer une première fois la mission. À l'aide d'algorithmes de tris, les étudiants réalisent une seconde solution plus efficace et entrevoient ainsi le principe de l'optimisation, non traité lors du TP 3 modifié. La dernière mission de la campagne est à réaliser en séance libre.

En raison du nombre important d'enseignants prenant part à cette formation (une quinzaine), une séance préalable leur a été proposée pour leur présenter le jeu, les modifications apportées aux sujets de TP et la solution des exercices. À l'issue de cette formation, de nombreuses remarques ont permis de faire évoluer les supports de TP avant la mise en œuvre auprès des étudiants. Par exemple : la séance de jeu, initialement prévue en encadrement lors de la première séance, est à réaliser en séance libre, d'où l'ajout d'une description détaillée en annexe du premier TP ; nous évitons désormais aux étudiants de chercher dans la bibliothèque en présentant chaque nouvelle fonction utile à la réalisation d'un exercice ; nous simplifions aussi le chargement de la bibliothèque en fournissant aux étudiants une base de fichiers préconfigurés.

Enfin, une ressource pédagogique a été créée pour permettre aux étudiants de préparer les TP chez eux. Cette initiative a été motivée par la difficulté d'installer le jeu notamment dans un environnement Linux. Pour simplifier cette procédure, un *live-DVD*[52] a été conçu et un exemplaire en a été distribuée à chaque étudiant. Il contient le jeu sérieux, les sujets de TP, divers langages de programmation et toutes les ressources nécessaires pour réaliser les TP. Son utilisation est simple, il suffit d'insérer le *live-DVD* dans un lecteur DVD et de redémarrer l'ordinateur. Celui-ci va alors amorcer le système d'exploitation inclus dans le *live-DVD*. Lorsque le chargement est terminé, l'étudiant peut utiliser le système pour jouer au jeu, réaliser les TP,

52. Un *live-DVD* contient en son sein un système d'exploitation complet ainsi qu'une suite de logiciels. Le *live-DVD* conçu à cette occasion est basé sur un système d'exploitation Linux (distribution Kubuntu).

écrire des rapports, préparer des présentations, etc. La difficulté principale liée à la conception de cette ressource repose sur l'exécution du jeu vidéo avec des performances graphiques correctes. En effet, le jeu sérieux étant basé sur *Kernel Panic*, il nécessite l'exploitation d'une carte graphique. Si celle-ci est mal gérée par le système, les performances du jeu sont grandement dégradées ce qui nuit à la jouabilité. Or, pour des raisons de portabilité, les systèmes inclus dans les *live-DVD* ne proposent qu'une gestion minimaliste des cartes graphiques ce qui rend souvent l'utilisation des jeux vidéo 3D impossible. Pour résoudre ce problème, les fichiers systèmes du *live-DVD* ont été modifiés pour détecter au démarrage la carte graphique présente sur l'ordinateur et installer le pilote associé s'il existe.

Ce *live-DVD* et la création de supports de TP propres au jeu sérieux sont les ressources qui permettent l'intégration du jeu dans la formation. Pour évaluer l'influence du jeu sur la motivation des étudiants et son utilisabilité par des enseignants externes au projet, un nouveau protocole d'évaluation s'impose.

### 5.3.3 Conception du protocole d'évaluation

Conformément aux objectifs fixés dans cette expérimentation, deux critères d'évaluation sont fixés : l'**influence du jeu sérieux sur la motivation des étudiants** ; et l'**appréciation du jeu sérieux par les enseignants**. De même que pour la première itération, une approche *ex-ante* est privilégiée et les indicateurs sont définis avant l'expérimentation de manière à être récoltés au fur et à mesure de son déroulement (voir figure 5.8).

Du point de vue de la motivation, l'objectif consiste à évaluer l'évolution de la motivation des étudiants entre le début et la fin du semestre. Pour ce faire, un questionnaire leur est distribué à deux reprises : la première fois au début du semestre, pour étalonner leur motivation ; la deuxième fois à la fin du semestre, pour permettre d'en constater les évolutions. Les questions sont donc légèrement remaniées en fonction de la première ou de la seconde distribution. Ces questionnaires sont basés sur le modèle de Viau [Via97] (présenté section 3.2.1). Chaque question fait référence à une composante de ce modèle dont les associations sont présentées dans le tableau 5.7. Les étudiants indiquent pour chaque question une valeur entre 1 et 7. La valeur maximale indique un acquiescement total à la question ou à l'affirmation alors que la valeur minimale dénote le contraire.

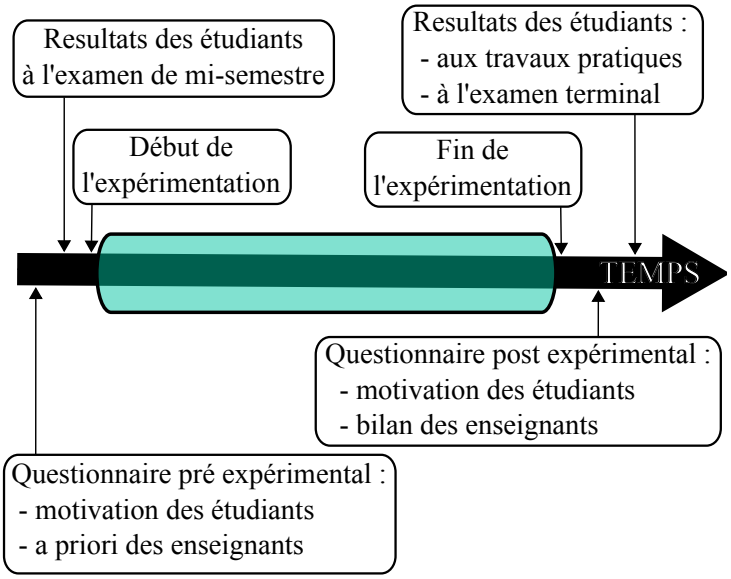



FIGURE 5.8 – Organisation temporelle de l'évaluation pour la troisième expérimentation.

TABLE 5.7 – Association entre les questions des questionnaires et les composantes du modèle motivationnel de Viau.

| Question | Composante motivationnelle | Poids |
|---|---|---|
| Avez-vous un projet professionnel ou de poursuite d'étude précis ? | **déterminant** (perception de la valeur d'une activité (perspective future)) | 2/48 |
| Ce que (j'apprendrai/j'apprends) en programmation me (valorisera/valorise) auprès de mon entourage. | **déterminant** (perception de la valeur d'une activité (buts sociaux)) | 2/48 |
| Je poursuis des études pour acquérir de nouvelles connaissances. | **déterminant** (perception de la valeur d'une activité (buts scolaires d'apprentissage)) | 2/48 |
| Je poursuis des études pour être estimé, reconnu ou encore pour obtenir un diplôme. | **déterminant** (perception de la valeur d'une activité (buts scolaires de performance)) | 2/48 |
| Je pense avoir les bases nécessaires pour poursuivre cette formation. | **déterminant** (perception de sa compétence à accomplir une activité (performances antérieures)) | 2/48 |

*Suite sur la prochaine page*

| Question | Composante motivationnelle | Poids |
|---|---|---|
| L'observation d'autres personnes réalisant des exercices me permet de me dire : « moi aussi je peux y arriver ». | **déterminant** (perception de sa compétence à accomplir une activité (observations)) | 2/48 |
| J'ai besoin que l'on me convainque de mes capacités pour accomplir une activité. | **déterminant** (perception de sa compétence à accomplir une activité (persuasion)) | 2/48 |
| Mon émotivité me rend incapable de réussir une activité comme, par exemple, un examen. | **déterminant** (perception de sa compétence à accomplir une activité (réactions physiologiques et émotives)) | 2/48 |
| Les difficultés que (je rencontre lors de mes études/j'ai rencontré en informatique) sont irrémédiables. | **déterminant** (perception de contrôlabilité (impuissance apprise)) | 2/48 |
| En général, mes résultats (/en programmation) sont plus liés à mes compétences qu'à des facteurs externes comme des grèves, les choix pédagogiques des enseignants, etc. | **déterminant** (perception de contrôlabilité (lieu de la cause : interne/externe)) | 2/48 |
| Mes résultats sont plus liés aux efforts que je fournis qu'à mes compétences intellectuelles. | **déterminant** (perception de contrôlabilité (stabilité de la cause : modifiable/stable)) | 2/48 |
| Mes échecs passés auraient pu être évités si je l'avais souhaité. | **déterminant** (perception de contrôlabilité (contrôle de la cause : contrôlable/incontrôlable) | 2/48 |
| Quand je suis en cours ou en TD et que l'enseignant parle, je pense à autre chose et je n'écoute pas tout ce qu'il dit. | **indicateur** (choix d'entreprendre une activité (évitement)) | 3/48 |
| Je rends toujours mes travaux à temps. | **indicateur** (choix d'entreprendre une activité (évitement)) | 3/48 |

*Suite sur la prochaine page*

| Question | Composante motivationnelle | Poids |
|---|---|---|
| Pour préparer les examens écrits, je fais toutes les annales des années précédentes. | **indicateur** (persévérance) | 3/48 |
| Je consacre une partie importante de mon temps à étudier. | **indicateur** (persévérance) | 3/48 |
| Lorsque j'étudie, j'utilise des techniques d'apprentissage comme, par exemple, réaliser des fiches, faire des schémas, etc. | **indicateur** (engagement cognitif (stratégies d'apprentissage)) | 3/48 |
| J'ai conscience des stratégies que j'utilise pour réguler ma façon de travailler. | **indicateur** (engagement cognitif (stratégies d'autorégulation : métacognitive)) | 1/48 |
| Je suis organisé dans mon travail universitaire. | **indicateur** (engagement cognitif (stratégies d'autorégulation : gestion)) | 1/48 |
| Quand je rencontre une activité difficile, je reste motivé pour l'accomplir. | **indicateur** (engagement cognitif (stratégies d'autorégulation : motivationnelle)) | 1/48 |
| Comment estimez-vous votre niveau en informatique. | **indicateur** (performance) | 3/48 |
| | Total | 45/48 |

Ce questionnaire permet d'évaluer chaque composante du modèle motivationnel de Viau, mais demande des données complémentaires telles que les notes des étudiants aux examens. Ces notes sont un indicateur de la performance des étudiants et sont à prendre en compte pour l'évaluation finale de l'évolution de leur motivation au cours du semestre. Avec un poids de $3/48$, cet indicateur complète donc le questionnaire présenté table 5.7.

Dans ce questionnaire, il n'y a pas de réponses justes ou fausses. Par exemple pour l'affirmation « *Je suis organisé dans mon travail scolaire* », l'évaluation ne consiste pas à déterminer si l'étudiant est effectivement organisé ou pas, mais simplement à observer comment il perçoit son organisation lors du premier questionnement (premier TP) puis lors du second (dernier TP). L'objectif est donc de récolter le point de vue de chaque étudiant avant et après l'expérimentation en vue d'observer une éventuelle évolution de sa motivation.

Du point de vue de l'encadrement des séances, cette expérimentation est la première à faire participer des enseignants externes au projet. À ce titre, deux questionnaires relativement ouverts ont été proposés aux enseignants. Le premier, distribué avant la première séance de travaux pratiques, interroge les enseignants sur leur rapport au jeu vidéo, leur pratique enseignante et leurs a priori sur le jeu sérieux proposé. Le deuxième questionnaire, distribué après la dernière séance de travaux pratiques, permet aux enseignants d'exprimer leur point de vue sur les difficultés rencontrées durant les séances, les conséquences du jeu sur les étudiants (motivation, apprentissage) et sur une éventuelle reconduite de l'expérimentation.

### 5.3.4 Résultats

#### Critère 1 : Influence du jeu sur la motivation des étudiants

L'évaluation de l'influence du jeu sérieux sur la motivation des étudiants est déterminée en grande partie par les réponses aux questionnaires distribués et par l'évolution de leurs résultats aux cours du semestre.

**Analyse des notes des étudiants :**

TABLE 5.8 – Calendrier des évaluations au L1S1 de l'Université Paul Sabatier.

| Semaine | TP | Évaluation |
|---|---|---|
| ... | | |
| 7 | | QCM3 |
| 8 | TP1 | Contrôle continu |
| 9 | TP2 | QCM4 |
| 10 | TP3 | |
| 11 | TP4 | |
| 12 | TP5 | QCM5 |
| 13 | TP6 | |
| ... | Révisions | |
| 17 | | Contrôle terminal |

L'obtention de l'unité d'enseignement dans laquelle le jeu sérieux a été intégré, dépend principalement de trois évaluations. Le calendrier de ces évaluations vis-à-vis des séances de TP est présenté dans le tableau 5.8. La première évaluation est un contrôle continu passé par les étudiants à mi-semestre, soit la même semaine que la première séance de TP. La deuxième

est la note de travaux pratiques attribuée en fonction de la participation des étudiants durant les TP. La troisième est un contrôle terminal de fin de semestre. Outre ces évaluations, un contrôle continu est mis en place sous la forme de QCM pour l'organisation du soutien. La conception de ces contrôles et l'évaluation des productions des étudiants sont réalisées par des enseignants externes au projet.

La figure 5.9 présente les moyennes des notes obtenues aux différentes évaluations (QCM exclus) en fonction des sous-groupes de TP. L'analyse de ces données fait apparaître une baisse des résultats entre le contrôle de mi-semestre et le contrôle terminal et cela quel que soit le sous-groupe de TP. Toutefois, cette baisse est moins importante pour les groupes ayant utilisé le jeu sérieux (« 1,25 » points contre « 1,55 » pour le groupe de référence).

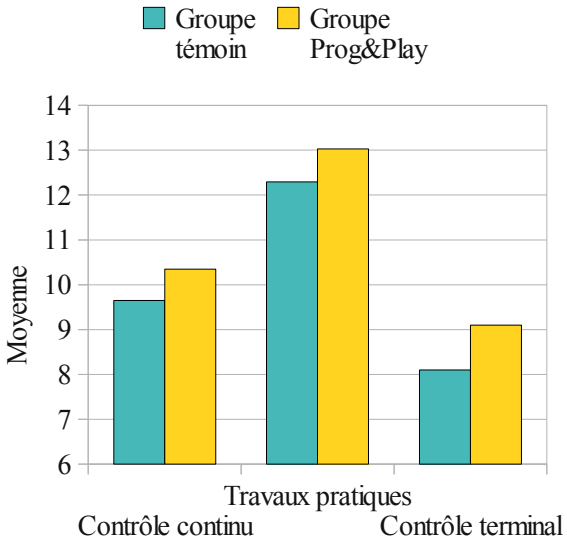



FIGURE 5.9 – Moyennes des notes obtenues aux évaluations en fonction du groupe d'étudiants.

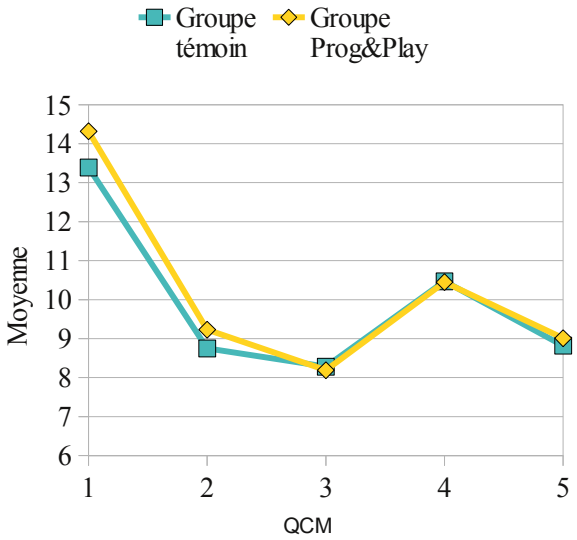



FIGURE 5.10 – Moyennes des notes obtenues aux QCM en fonction du groupe d'étudiants.

Cette différence ne peut être attribuée qu'à la seule utilisation du jeu. En effet, de nombreux paramètres incontrôlable, dû à la mise en œuvre de l'expérimentation en contexte réel, ont également pu participer à ce résultat. Par exemple, les étudiants ayant utilisé le jeu étaient en moyenne plus performants que les autres avant même le début des TP (voir moyennes du contrôle continu figure 5.9). Cette différence initiale doit être prise en compte pour interpréter les résultats. Néanmoins, l'écart initial constaté lors du contrôle continu (« 0,7 » point) est maintenu

lors de l'évaluation de TP (« 0,74 » point) et augmenté lors du contrôle terminal (« 1 » point). Le jeu sérieux est donc l'un des nombreux facteurs ayant contribués à obtenir ces résultats.

Concernant l'analyse des QCM (voir figure 5.10), la meilleure performance initiale des étudiants du groupe Prog&Play est confirmée lors du premier QCM. Toutefois, les résultats entre les deux groupes se réduisent lors du deuxième QCM et deviennent équivalents à partir du troisième. Il faut noter que le contrôle continu a été réalisé la semaine suivant le QCM 3 et que les séances de TP avec le jeu ont eu lieu entre les QCM 4 et 5 (voir tableau 5.8). Par conséquent, les résultats au QCM numéro 5, centré sur les listes, apportent l'information selon laquelle le TP4 basé sur le jeu sérieux a tout aussi bien préparé les étudiants à la manipulation des listes que le TP classique.

Conformément au protocole d'évaluation, la variation des notes entre le contrôle continu et le contrôle terminal de chaque étudiant est intégrée au calcul de la motivation.

**Évolution de la motivation :** L'analyse de cette évolution est conduite suite aux questionnaires retournés par les étudiants. Parmi les trois cents quatre étudiants inscrits pédagogiquement, cent soixante-seize ont retourné les deux questionnaires dont quatre-vingt six appartenant au groupe témoins et quatre vingt dix au groupe Prog&Play ; l'analyse des données s'effectue donc sur cet échantillon.

Le tableau 5.9 présente l'évolution moyenne de la motivation en fonction des groupes de TP. Le premier constat porte sur la motivation générale qui est restée relativement stable malgré une légère baisse quel que soit le groupe considéré. Les étudiants du groupe 2 semblent avoir tout de même mieux conservé leur motivation (baisse de « 0,03 » points) par rapport aux étudiants du groupe 1 (baisse de « 0,21 » points). La différence entre ces deux groupes est de l'ordre de « 0,18 » point sur l'échelle de valeur comprise entre 1 et 7. On peut noter que 46% des étudiants du groupe 2 ont vu leur motivation progresser contre 35% du groupe 1.

TABLE 5.9 – Évolution moyenne de la motivation en fonction des groupes de TP.

| **Groupe numéro** | **Utilisation du jeu** | | |
|---|---|---|---|
| | | **Motivation** | |
| 1 | | -0,21 | |
| 2 | X | -0,03 | |
| | | **0,18** | **Différence** |

Pour déterminer la composante principalement responsable de cette légère variation, l'analyse est portée au niveau des déterminants et des indicateurs (voir tableau 5.10). Il apparaît ainsi que cette variation est répartie sur l'ensemble des composantes du modèle motivationnel de Viau à l'exception de la performance (I4) qui est identique entre les deux échantillons.

TABLE 5.10 – Évolution moyenne de la motivation pour chaque composante du modèle motivationnel de Viau en fonction des groupes de TP.

| **Groupe numéro** | **Utilisation du jeu** | **Déterminant** | | | **Indicateur** | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | **D1** | **D2** | **D3** | **I1** | **I2** | **I3** | **I4** | |
| 1 | | -0,25 | -0,13 | -0,29 | -0,35 | -0,25 | -0,07 | -0,1 | |
| 2 | X | -0,12 | 0,09 | -0,09 | -0,03 | -0,07 | 0,05 | -0,07 | |
| | | **0,13** | **0,23** | **0,20** | **0,32** | **0,18** | **0,12** | **0,03** | **Différence** |

**D1** : Perception de la valeur d'une activité
**D2** : Perception de sa compétence à accomplir une activité
**D3** : Perception de la contrôlabilité d'une activité
**I1** : Choix d'entreprendre une activité
**I2** : Persévérance
**I3** : Engagement cognitif
**I4** : Performance

Un dernier indice de motivation concerne le choix d'orientation des étudiants au second semestre. En effet, suite au parcours IMP (Informatique, Mathématiques, Physique) suivi au premier semestre, les étudiants ont le choix au second semestre entre neuf majeures dont celle incluant l'informatique (IMM - Informatique, Mathématiques, Mécanique). Parmi les étudiants du groupe Prog&Play, 55% ont choisi cette majeure contre 49% pour le groupe témoin.

À la vue de ces résultats, il convient de rester prudent quant aux conclusions trop hâtives. En effet, suite à l'enquête et au suivi des étudiants, les données récoltées plaident en faveur du jeu sérieux pour maintenir la motivation. Toutefois, les différences enregistrées restent relativement faibles et nous ne pouvons affirmer que les différences observées sont exclusivement dues à l'intégration du jeu sérieux dans la formation.

Pour compléter ces résultats, le point de vue des étudiants sur le jeu et cette expérimentation a été récolté lors de l'enquête post expérimentale. Concernant l'amusement, les deux questions présentées dans la figure 5.4 (page 130) ont été réutilisées. La réponse moyenne pour la première question est de « 4,14 » soit une appréciation modérée du scénario du jeu. Cette impression se retrouve naturellement dans les réponses à la seconde question. Alors que 100% des étudiants de la première expérimentation avaient trouvé que l'utilisation de la programmation dans le jeu

augmentait le divertissement, 42% des étudiants de cette enquête votent pour une augmentation, 22% n'indiquent aucun changement, 21% ne se prononcent pas et 15% trouvent que cela le réduit. Donc 64% des étudiants estiment que le jeu sérieux est au moins aussi divertissant que le jeu sans la programmation.

Pour la pyramide des besoins (voir figure 5.11), les trois premiers niveaux de base restent relativement hauts alors que les niveaux 4, 5 et 7 ont fortement diminués par rapport à la première expérimentation (voir figure 5.5 page 131). Alors que le jeu était identique à la première itération (à l'exception de la nouvelle mission), les résultats sur l'utilisabilité et l'amusement sont inférieurs. Pour expliquer cette différence, les critiques des étudiants ont été étudiées et les remarques récurrentes portent essentiellement sur la lourdeur des supports de TP. Nous inférons de ces critiques que les supports de TP ont nui à la compréhension du jeu et à son potentiel ludique.

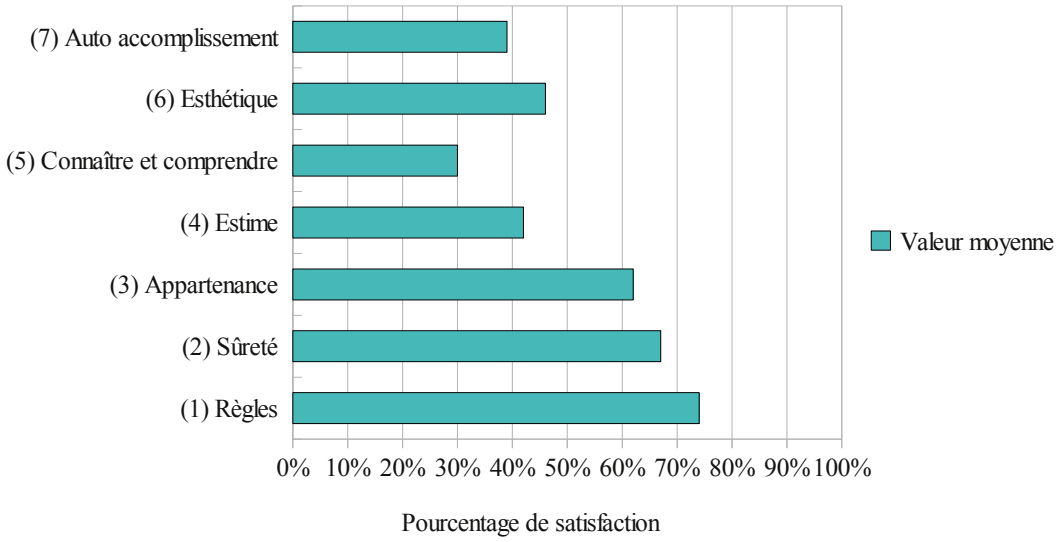



FIGURE 5.11 – Satisfaction moyenne des étudiants pour chaque niveau de la pyramide des besoins (L1S1 IMP Toulouse III).

Malgré cela, les étudiants ont dans l'ensemble apprécié l'initiative et sont demandeurs de ce type de pédagogie (voir figure 5.12). 89% des étudiants interrogés n'ont pas d'opinion défavorable sur l'intérêt d'utiliser un jeu vidéo pour apprendre à programmer et 85% pensent qu'un TP basé sur un jeu vidéo peut avoir sa place dans cette formation.

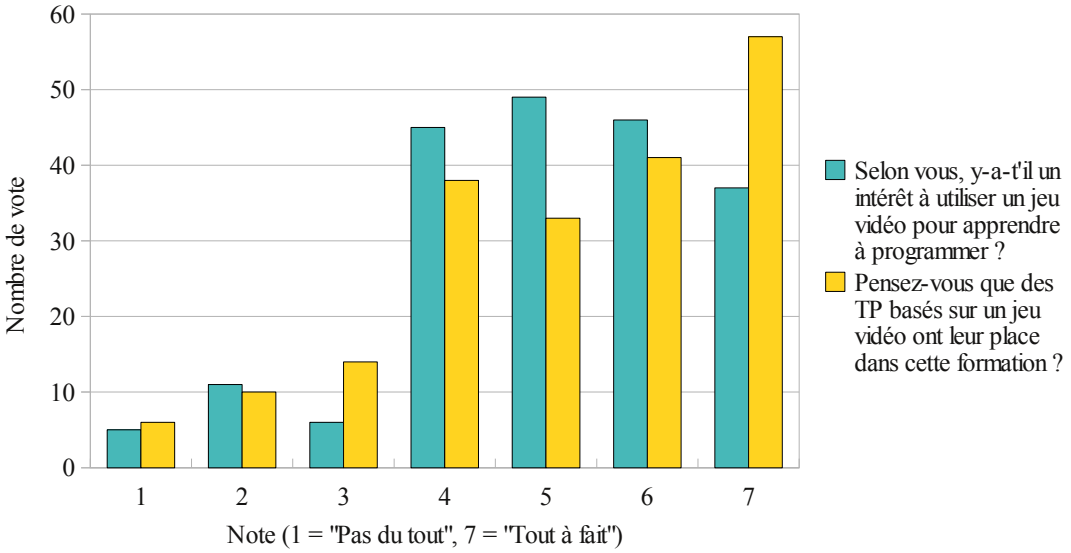



FIGURE 5.12 – Perception des étudiants de l'utilité d'un jeu vidéo pour apprendre la programmation.

### Critère 2 : Appréciation du jeu sérieux par les enseignants

Pour évaluer l'appréciation du jeu sérieux par les enseignants, deux questionnaires ont été réalisés. Le premier a été distribué lors de la phase de définition du protocole d'évaluation durant le second semestre de l'année universitaire précédent cette itération. Le deuxième a été rempli par les enseignants de TP à la fin du premier semestre de l'année universitaire contemporaine à l'expérimentation. En raison de ce chevauchement, certains enseignants ayant collaborés à l'intégration du jeu sérieux dans la formation n'ont pas participé à l'expérimentation et vice versa. Par conséquent, la population des enseignants est légèrement différente pour les deux questionnaires.

Pour la première enquête, quinze enseignants ont répondu dont trois femmes et douze hommes. Dans cet échantillon, neuf attestent avoir une expérience de joueur. Ce premier questionnaire les interroge sur les jeux vidéo, leur pratique enseignante et leur a priori sur le concept d'un jeu sérieux appliqué à la programmation.

**Quels sont, selon vous, les qualités et défauts inhérents aux jeux vidéo ?** À cette question, les enseignants ont principalement relevé trois qualités dont la capacité des jeux vidéo à favoriser la réflexion à travers le développement de stratégies de jeu, à détendre et à permettre

l'immersion dans des environnements virtuels. Dans une moindre mesure, l'aspect compétition à travers le mode de jeu multijoueur, le scénario, la motivation et la créativité sont notés comme étant des qualités intrinsèques aux jeux vidéo. Concernant les défauts, la principale inquiétude porte sur le danger à l'addiction et à la coupure de la réalité. Certains enseignants les trouvent complexes et les associent à une perte de temps.

**Quels problèmes rencontrent vos étudiants dans l'apprentissage de la programmation et quelles solutions mettez-vous en œuvre pour tenter d'y répondre ?** Selon les enseignants interrogés, les étudiants se trouvent principalement en difficulté pour exprimer une solution algorithmique à un problème donné. Cette difficulté est en partie due, selon eux, à des lacunes dans la maîtrise de l'algorithmique et des structures de contrôle et à des difficultés de raisonnement logique et d'abstraction. À cela, s'ajoute un manque de motivation, de travail et de rigueur, associé aux problèmes de syntaxe inhérents aux langages de programmation. Pour aider les étudiants à surmonter ces difficultés, les enseignants utilisent majoritairement l'exemple et suggèrent des méthodes de travail pour stimuler l'autonomie des étudiants.

**A priori, pensez-vous qu'un jeu vidéo puisse aider les étudiants à surmonter leurs difficultés ? Pourquoi ?** Pour cette question, six enseignants ne se prononcent pas, sept portent un jugement positif et deux négatif. Les a priori positifs positionnent le jeu vidéo comme un outil concret, proche du centre d'intérêt des étudiants qui permet une visualisation immédiate de leurs programmes dans le contexte du jeu. Mais la principale hypothèse positive porte sur la capacité du jeu vidéo à augmenter la motivation des étudiants à travailler en informatique. Les quelques a priori négatifs se fondent sur la crainte : de donner une image erronée de l'informatique qui ne se cantonne pas aux jeux vidéo ; de favoriser principalement les garçons ; de perdre du temps à expliquer le jeu au détriment des concepts algorithmiques ; et de détourner les étudiants du fond en raison de la dimension ludique du jeu. Les enseignants sans opinion sont en attente de résultats.

**Etes vous favorable à cette expérimentation ? Pourquoi ?** Cette question était fondamentale pour la mise en place du jeu sérieux dans la formation. Il n'était pas envisageable de déployer le jeu si les enseignants de travaux pratiques n'avaient pas été consentants. Sur les quinze questionnaires récoltés, onze sont favorables à la mise en place de l'expérience et quatre ne se sont pas prononcés. Les raisons de cette acceptation sont principalement motivées par le souhait

de faire évoluer l'offre de formation, par l'espoir de motiver les étudiants et par la simple curiosité des résultats à venir.

Pour la seconde enquête, quinze enseignants dont douze identiques au premier questionnaire ont répondu. Ce groupe se compose de quatre femmes et onze hommes. Ce second questionnaire permet aux enseignants de dresser un bilan de l'expérimentation. Concernant la préparation des séances de TP basées sur le jeu sérieux, l'ensemble des enseignants affirme que cela demande une charge de travail équivalente à la préparation d'un nouveau TP. Néanmoins, les enseignants non joueurs ont particulièrement apprécié la séance de formation de quatre heures organisée avant le premier TP.

Du point de vue des conséquences du jeu sur la motivation et les performances des étudiants, le bilan est fortement contrasté. Sur le plan des performances, la plupart des enseignants ont le sentiment que les TP modifiés ont été contre-productifs pour de nombreux étudiants (il faut noter que se ressenti ne se retrouve pas dans les résultats aux évaluations présentées figures 5.9 et 5.10 page 145). Les principales critiques portent essentiellement sur les supports de TP trop longs, d'où une perte de temps et une pratique de la programmation moins importante que pour les TP classiques. En revanche sur le plan de la motivation, la majorité des enseignants constate un effet positif sur l'essentiel des étudiants.

Enfin il leur a été demandé leur point de vue quant à une éventuelle reconduite de l'expérimentation l'année suivante. Deux enseignants ne souhaitent pas voir cette expérimentation reconduite. Le premier qui a uniquement encadré un TP classique s'oppose simplement au principe d'intégrer un jeu dans la formation. Le second nuance son opinion, il trouve que l'API Prog&Play complique les TP ce qui désavantage les étudiants déjà en difficulté. Cependant, il ajoute que s'il existe une solution permettant de simplifier l'utilisation de la bibliothèque et la manipulation des entités dans le jeu, ce type de TP pourrait être maintenu et généralisé.

Les autres enseignants sont favorables à une reconduite de l'expérimentation sous réserve de modifications. Six d'entre eux proposent de conserver les deux formes de TP mais suggèrent de laisser le choix aux étudiants. Sept d'entre eux préfèreraient généraliser les TP basés sur le jeu de façon à fournir une formation équivalente à tous les étudiants. Dans tous les cas, ces enseignants sont en demande d'une simplification de la bibliothèque et des sujets de TP.

**Synthèse de la troisième expérimentation**

Cette expérimentation est donc riche en enseignements et a permis de mettre en évidence la difficulté de mener une expérimentation dans le cadre d'une formation de plusieurs centaines d'étudiants. En effet en raison du nombre important d'étudiants, l'intégration du jeu a dû être conduite dans un cadre moins souple que la réalisation d'un atelier ou d'un soutien. Le jeu a donc été intégré dans le calendrier des enseignements et les TP ont été adaptés pour conserver l'ensemble des concepts manipulés lors des TP classiques. La conséquence de cette fusion a rendu les sujets proposés difficilement réalisables dans le temps imparti.

Ainsi, cette expérimentation met en évidence que les résultats obtenus dépendent non seulement du jeu sérieux mais aussi de son déploiement, de l'encadrement, des documents, etc. Cette première expérimentation à grande échelle a donc permis de valider la portabilité du jeu avec une approche fonctionnelle et d'éclaircir les préconditions au déploiement du jeu (formation des enseignants, rédaction des supports d'enseignements et adaptation des calendriers). Fort de cette expérience, les prochaines expérimentations de cette ampleur devront intégrer de manière plus importante le jeu dans la formation. À titre d'exemple, il serait envisageable d'étudier les missions de la campagne en TD à travers la présentation de l'API et l'étude des algorithmes de chaque mission. L'ajout de séances de TP supplémentaires dédiées au jeu pourrait être une solution pour permettre une intégration naturelle du jeu sans dénaturer le contenu fondamental de la discipline ou réduire son niveau académique. En effet, lors de cette expérimentation, la part du jeu dans la formation n'a représenté que quatre heures de travaux pratiques sur les soixante douze heures que compte l'unité d'enseignement. Laisser une part plus importante au jeu sérieux devrait permettre d'aborder les TP de manière plus progressive de façon à permettre aux étudiants de conserver une certaine initiative. De cette manière, la composante « jeu » sera révélée et favorisera le développement de la motivation.

## 5.4 Diffusion du jeu sérieux

Suite à la réalisation de ces trois expérimentations et à la communication réalisée autour de ce projet, quelques collaborations ont pu être initiées et ont permis de déployer le jeu dans de nouvelles formations. Pour aider les enseignants intéressés par le projet, un protocole d'évaluation générique a été conçu pour servir de cadre modulaire au déploiement du jeu. Ce protocole est inspiré de ceux mis en place lors des premières expérimentations.

La principale nouveauté porte sur l'ajout d'une troisième phase de manière à illustrer la possibilité d'utiliser le mode de jeu multijoueur (voir tableau 5.11). Cette dernière phase peut être abordée à travers une approche par projet, où les étudiants développent en autonomie leur propre IA. L'ensemble des solutions présentées page 108 peuvent être déployées lors de cette phase. Si les étudiants sont autorisés à communiquer entre eux, ils peuvent développer des stratégies d'alliance ou de coopération, ou simplement s'entraider sur des problèmes de programmation. Lorsque tous les programmes sont complets et opérationnels, la compétition peut être organisée pour permettre aux étudiants de jouer en utilisant leurs programmes. Une attention particulière doit être apportée au rôle des programmes, à l'activité des étudiants et aux stratégies utilisées. Ces observations sont la base de la dernière étape où les étudiants et les enseignants analysent le jeu et tentent de trouver les raisons des victoires et des défaites.

TABLE 5.11 – Organisation des enseignements avec prise en compte du mode multijoueur.

<table>
<tr><th>Phase 1</th><th>Phase 2</th><th>Phase 3</th></tr>
<tr><td>Présentation du jeu initial et jeu en réseau</td><td>Présentation de l'API Prog&Play et réalisation des missions de la campagne<table><tr><th>Pour chaque mission</th></tr><tr><td>(1) Présentation des objectifs<br>(2) Les étudiants programment leur(s) solution(s)<br>(3) Institutionnalisation</td></tr></table></td><td>Réalisation d'IA et test des programmes créés lors de parties en mode multijoueur</td></tr>
</table>

Outre cette proposition d'organisation des enseignements, les questionnaires et enquêtes relatives au jeu sérieux sont mises à disposition des enseignants. De nombreuses informations et documents d'aide sur l'installation du jeu et son utilisation sont également disponibles sur les pages web associées au jeu sérieux [53].

Les expérimentations présentées ci-dessous sont donc les illustrations de la diffusion du jeu sérieux dans diverses formations.

53. `http://www.irit.fr/~Mathieu.Muratet/progAndPlay.php` accédé le 07 Août 2010

### 5.4.1 Première diffusion

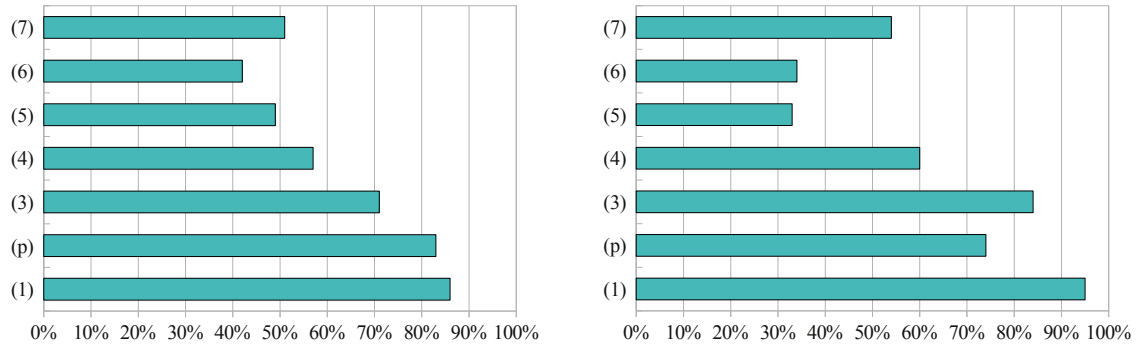
Cette expérimentation a été menée par André Péninou et Marie-Françoise Canut du département informatique de l'IUT B de Blagnac (Toulouse II). Le jeu sérieux a été déployé dans deux unités d'enseignement différentes. La première mise en œuvre a eu lieu au premier semestre sous la forme d'un soutien à l'apprentissage du langage C. Trente cinq étudiants, sur les cent quinze de la promotion, ont participé à trois séances d'une heure et trente minutes. Ces étudiants ont été sélectionnés en fonction de leur résultat au contrôle continu d'algorithmique (note inférieure à 10/20). L'objectif de ce soutien était de leur faire approfondir les éléments fondamentaux du langage et les sous-programmes.

La seconde mise en œuvre a eu lieu au semestre 3 sous la forme d'un soutien à l'apprentissage du langage Java. Une quinzaine d'étudiants volontaires, sur les cent huit de la promotion, ont participé aux trois séances d'une heure et trente minutes. L'objectif de ce soutien était de faire manipuler le concept d'objet à travers le jeu. De part ce déploiement, ces enseignants ont montré la portabilité du jeu dans un module de programmation orienté objet. Les étudiants étaient utilisateurs du diagramme de classe présenté figure 4.25 page 115.

Concernant le ressenti des enseignants sur ces expérimentations, le point positif souligné porte sur la mise en pratique originale de la programmation (contexte différent des enseignements classiques, retour immédiat des programmes exécutés). À l'inverse, une première critique porte sur la richesse de l'API, difficile à maîtriser par les étudiants en seulement trois séances. Une adaptation plus approfondie aurait dû être apportée notamment pour la première mise en œuvre. Une seconde limite concerne la nécessité, pour l'enseignant, de maîtriser le jeu afin de comprendre les différentes stratégies qui peuvent amener aux solutions de chaque mission. Cette difficulté rencontrée par les enseignants se retrouve dans l'analyse de la pyramide des besoins du joueur où la satisfaction des niveaux supérieurs est relativement moyenne alors que les bases sont particulièrement bien posées (voir figures 5.13a et 5.13b).

Du point de vue de la motivation, l'analyse des questionnaires récoltés lors de la première mise en œuvre (quarante-trois au total dont vingt-trois étudiants ayant participé au soutien) montrent une augmentation de la motivation de « 0,43 » point pour les étudiants ayant participé au soutien et une augmentation de « 0,21 » point pour les autres.

Enfin, sur les deux expérimentations, le principe du jeu sérieux semble être apprécié par les étudiants car 55% des étudiants ont trouvé que l'introduction de la programmation augmentait le divertissement, 29% ont trouvé que l'amusement était identique et seulement 8% ont trouvé




(a) Semestre 1 - langage C.

(b) Semestre 3 - langage Java.

(7) Auto accomplissement
(6) Esthétique
(5) Connaître et comprendre
(4) Estime
(3) Appartenance
(2) Sûreté
(1) Règles

FIGURE 5.13 – Satisfaction moyenne des étudiants pour chaque niveau de la pyramide des besoins (IUT B Toulouse II).

que cela le réduisait (8% ne se prononcent pas). Donc, 84% des étudiants estiment que le jeu sérieux est au moins aussi divertissant que le jeu sans la programmation.

### 5.4.2 Deuxième diffusion

Suite à la première expérimentation au département informatique de l'IUT A (Toulouse III), les ressources créées sur la plateforme Moodle lors de cet enseignement ont été restructurées pour créer un squelette de cours utilisable par l'ensemble des départements de l'IUT. Cet espace est organisé autour de trois thèmes : la présentation du jeu, l'ingénierie pédagogique associée et les aspects techniques. Il peut être exporté par les enseignants pour leur propre utilisation et leur fournit un cadre pédagogique directement opérationnel. L'enseignant adapte alors le jeu à son contexte pédagogique et peut, à son tour, distribuer ces contributions et ainsi participer au développement du jeu sérieux.

Cette initiative a permis de déployer le jeu dans deux nouveaux départements : SeRéCom (Services et Réseaux de Communication) du site Castrais et GE2I (Génie Electrique et Informatique Industrielle) du site Toulousain.

Concernant le département SeRéCom, les enseignants (Sylvain Barreau et Emmanuel Conchon) ont intégré le jeu à la formation initiale du premier semestre dans l'unité d'enseignement « culture technologique et développement multimédia – initiation à la programmation ». Soixante étudiants, répartis dans quatre groupes de TP, ont alors utilisé le langage C pour réaliser la campagne du jeu sur un volume horaire d'une dizaine d'heures. Ce module vise l'apprentissage des structures algorithmiques de base et de la modularité via les fonctions. Suite à cette première utilisation, ils envisagent de reconduire l'expérience avec la même organisation pédagogique (tous les étudiants, premier semestre, module Algorithmique). Toutefois, ils notent la nécessité de mieux préparer les séances, ils proposent notamment d'adapter la bibliothèque au niveau de leurs étudiants et d'intégrer plus étroitement le jeu au reste de la formation et à l'évaluation du module dans lequel il est utilisé.

Pour le second département (GE2I), le jeu a été utilisé dans une filière dite de « remédiation » comprenant des étudiants en difficulté. Cette filière propose un aménagement des enseignements destiné à permettre aux étudiants de raccrocher la formation initiale sans subir un redoublement. Dans ce contexte, l'expérimentation était centrée sur la capacité du jeu à motiver les étudiants à pratiquer la programmation. Sur un volume horaire de 10 heures, quinze étudiants ont utilisé le C++ pour résoudre la campagne du jeu. Ces enseignements ont été conduits par Jérémie Guiochet qui souhaite également renouveler l'expérience. Selon lui, les étudiants ont montré leur implication en travaillant chez eux alors que cela n'était pas demandé. Il note également que le profil des « remédiés » est particulièrement bien adapté à ce projet. Par ailleurs, Monsieur Guiochet a notamment pu mettre en place avec quelques étudiants la troisième phase de l'organisation des enseignements présentée tableau 5.11 page 153. Lors de cette phase, les étudiants ont adapté leur algorithme de la dernière mission pour le tester dans une session en mode multijoueur. Enfin, Monsieur Guiochet a utilisé le squelette de scénarios pour créer un contexte de jeu original utilisé pour l'évaluation des TP. Dans cet examen, l'étudiant contrôle une armée affaiblie composée d'un Assembleur, d'un Byte et de neuf Bits pour combattre une douzaine de Bits adverses. Les questions posées traitent de la conception de fonctions et de la manipulation de tableaux. Les étudiants doivent notamment rechercher l'assembleur parmi l'ensemble de ses unités, trier les unités endommagées par capital santé croissant et les réparer, rechercher la position de l'armée adverse sur la carte et lancer une attaque.

### 5.4.3 Troisième diffusion

Une dernière collaboration a été menée avec Elisabeth Delozanne et Françoise Le Calvez de l'université Pierre et Marie Curie (Paris VI). À cette occasion, une nouvelle mission a été ajoutée à la campagne qui se compose donc des sept missions présentées page 106. L'ajout de cette nouvelle mission, par rapport aux autres expérimentations, a pour objectif de décomposer le problème de la réparation (Mission 4). En effet, cet exercice est perçu par les étudiants comme étant le plus complexe, car il aborde deux difficultés : la recherche d'un élément dans un ensemble ordonné et l'application d'une opération à chaque élément de cet ensemble. Le problème de la recherche est donc abordé une première fois dans la nouvelle mission 5 où le joueur doit rechercher l'assembleur parmi l'ensemble des unités et le déplacer à la rencontre de l'armée. La mission 6 (anciennement numéro 5) peut alors être considérée plus sereinement de par le traitement de la recherche lors de la mission précédente. La dernière mission a également été mise à jour afin de proposer un défi plus conséquent pour terminer la campagne. En effet, au cours des différentes expérimentations, de nombreux étudiants ont été frustrés par sa simplicité.

Une première expérimentation a été menée avec 5 enseignants (deux en école primaire et trois en lycée) qui suivent un master 2 spécialité « Ingénierie de la formation en ligne » dans lequel ils ont choisi une unité d'enseignement libre de Java. Le jeu a été utilisé durant deux séances de trois heures à une semaine d'intervalle. L'objectif de cette expérimentation était de prendre un premier contact avec le jeu dans un contexte d'enseignement très souple. Les difficultés rencontrées étaient principalement d'ordre technique (installation du jeu sur les ordinateurs portables des participants et la configuration du réseau). Les participants à l'expérimentation auraient souhaité un environnement simple pour installer et configurer le jeu automatiquement sur leur ordinateur. Toutefois, en raison de l'orientation modulaire du jeu sérieux, il semble difficile de fournir un environnement compatible avec l'ensemble des mises en œuvre possibles. Par conséquent, il incombe aux enseignants et au service technique associé à la formation d'installer le jeu et de configurer le système pour utiliser l'environnement de programmation de la formation et permettre le jeu en réseau si besoin est.

Suite à ce premier test, une deuxième expérimentation a eu lieu dans le cadre d'une unité d'enseignement transverse orientée TICE[54] du L1 (intitulée « Insertion TICE »). Dans cette UE, l'enseignant propose un ensemble de projets dont un était basé sur le jeu sérieux. Il a donc demandé à Elisabeth Delozanne et Françoise Le Calvez de prendre en charge quelques étu-

54. TICE : Technologies de l'Information et de la Communication dans l'Enseignement

diants pour les faire travailler sur la résolution de la campagne à l'aide du langage C. Parmi l'ensemble des volontaires, huit étudiants qui suivaient le module d'initiation à la programmation impérative en C au second semestre ont été retenus.

Deux séances de deux heures en présentiel ont été organisées à une semaine d'intervalle pour présenter le jeu et les éléments de programmation non traités en cours (utilisation d'une bibliothèque, enregistrements, etc.). Suite à cette présentation, les trois premières missions ont été étudiées d'abord à l'aide du langage algorithmique conçu lors de la deuxième expérimentation (voir annexe D) puis en langage C. Le travail en binôme est encouragé par les encadrantes qui notent un réel intérêt des étudiants pour le jeu lui-même et pour l'interface de la bibliothèque. Suite à ces deux séances, les binômes avaient deux mois pour terminer la campagne en autonomie et préparer une présentation sur le travail effectué. À plusieurs reprises des rendez-vous étaient proposés aux étudiants pour les aider dans les difficultés rencontrées, seul un étudiant a jugé bon d'en profiter. Le jour de la soutenance, six étudiants sur huit étaient présents. Deux étudiants ont travaillé seuls et les quatre autres en binôme. Les deux étudiants absents ont demandé à passer la deuxième session.

Le bilan dressé par Elisabeth Delozanne et Françoise Le Calvez est très positif. Elles ont d'abord été impressionnées par la diversité des stratégies envisagées et des exposés préparés. Les étudiants ont trouvé qu'ils programmaient quelque chose de réel et ont apprécié ce projet. Les cinq étudiants qui ont terminé la dernière mission ont spontanément voulu tester la stratégie développée dans un contexte de jeu réel. La motivation était donc largement au rendez-vous.

Ce type de projet sera donc reconduit avec les adaptations suivantes : mettre en place une session de jeu en réseau ; recommander l'écriture de fonctions pour décomposer chaque problème en sous problèmes ; avant la soutenance organiser une séance de regroupement pour faire le point, dépanner ceux qui sont bloqués et permettre à ceux qui ont fini, de tester leur stratégie contre d'autres binômes ; inciter les étudiants à communiquer par courriel et via la plateforme d'enseignement Sakai.

## Synthèse des diffusions

Lors de ces collaborations, sept enseignants ont utilisé le jeu dans leurs formations. Ces quelques expérimentations supplémentaires, réalisées pour certaines en totale autonomie, illustrent la portabilité du jeu sérieux dans différents contextes d'enseignements. De part la dé-

marche volontaire de ces enseignants, ils ont su prendre le temps de maîtriser l'outil et de l'intégrer dans leur projet pédagogique.

Concernant les difficultés rencontrées par les enseignants lors du déploiement du jeu, elles sont essentiellement d'ordre technique (installation et paramétrage du jeu et configuration du réseau pour autoriser les parties en mode multijoueur). Du point de vue de la conduite des séances, les premières mises en œuvre permettent aux enseignants de réaliser qu'il est nécessaire de préparer le jeu pour leurs étudiants. En effet, le jeu tel qu'il est distribué fournit une base pour son déploiement mais demande à être adapté à chaque contexte d'enseignement. Cette préparation peut passer par une simplification des interfaces de l'API, une adaptation du calendrier des enseignements, l'ajout de nouvelles missions, etc.

# Conclusion

Ces quelques expérimentations fournissent un premier retour sur le jeu sérieux. Le cadre fourni par l'ingénierie didactique de Cobb *et al.* [CCD+03] a permis de réaliser trois itérations. L'objectif de la première expérimentation consistait à observer une première confrontation entre des étudiants et le jeu. L'évaluation a donc porté sur les composantes fondamentales de ce jeu sérieux : l'apprentissage, l'amusement et l'utilisabilité. Grâce aux données récoltées, l'amusement et l'utilisabilité du jeu sérieux ont été validés. Concernant l'apprentissage, des données supplémentaires étaient requises pour évaluer l'impact du jeu sur cette composante.

Suite à cette première expérimentation, la question de la portabilité du jeu dans un contexte d'enseignement différent s'est alors posée. La deuxième itération s'est donc centrée sur cette problématique. À cette occasion l'observation d'une séance d'enseignement a permis de récolter des informations sur l'influence de l'intégration du jeu sur les activités de l'enseignant.

Après avoir vérifié la portabilité du jeu lors de la deuxième expérimentation, l'itération suivante a consisté à déployer le jeu à plus grande échelle afin d'évaluer l'influence de ce dernier sur la motivation des étudiants et son appréciation par des enseignants externes au projet. Suites aux données récoltées, trois préconditions aux déploiement du jeu ont pu être dégagées : former les enseignants à l'utilisation du jeu sérieux, prévoir une place suffisante pour le jeu sérieux dans le calendrier des enseignements et concevoir les ressources d'enseignement en conséquence.

Entre ces trois itérations et la diffusion du jeu, plus de trois cents étudiants ont manipulé le jeu sérieux et une vingtaine d'enseignants externes au projet ont pris part à l'encadrement de séances. Grâce à l'ensemble des expérimentations et des commentaires laissés par tous ces

participants, le jeux sérieux composé de la campagne présentée page 106 semble être plus particulièrement adapté à un public d'étudiants volontaires où l'encadrement laisse une place suffisante au jeu.

Le détail des données relatives à ces expérimentations peut m'être demandé à cette adresse : Mathieu.Muratet@irit.fr.

# Conclusion des contributions

Le travail effectué dans cette thèse a donc consisté à concevoir, réaliser et évaluer un jeu sérieux pour l'apprentissage des fondamentaux de la programmation.

Concernant la conception, les fondements du jeu sérieux sont structurés autour des modes de jeu campagne et escarmouche où l'API Prog&Play permet aux étudiants d'interagir avec le jeu par la programmation. Le mode campagne est utilisé pour proposer aux étudiants un scénario original composé de missions à difficulté croissante. Le mode escarmouche incite les étudiants à travailler sur des projets où l'objectif final consiste à relever un défi pour remporter une compétition finale. Dans les deux cas, les missions ou les projets posent des problèmes dans le contexte du jeu qui doivent être résolus par la réalisation de programmes informatiques.

Ce jeu sérieux a été réalisé à l'aide du moteur de jeu de STR *Spring* et du jeu *Kernel Panic*. Dans cet univers, une campagne composée initialement de cinq missions a évolué au cours des expérimentations jusqu'à atteindre les sept missions actuelles. Le choix d'un jeu de STR comme support au jeu sérieux est justifié par trois composantes : c'est un environnement complexe propice à la mise en place de problèmes informatique, c'est un type de jeu populaire auprès des étudiants et de nombreux moteurs à code source ouvert sont accessibles sur Internet. L'interaction avec le jeu par la programmation est rendue possible à l'aide de l'API Prog&Play. Cette API est toujours en évolution de manière à proposer une interface la plus simple possible tout en conservant une interaction maximale avec des jeux de STR différents. Une part importante des spécifications du jeu sérieux porte sur le problème de sa portabilité à différents contextes d'enseignement. Dans ce cadre, l'interface « Client » de l'API Prog&Play a été traduite dans différents langages de programmation adapté aux approches d'enseignement impérative, orientée objet et fonctionnelle.

Grâce à cette ouverture, le jeu sérieux a pu être déployé dans de nombreux contextes d'enseignements différents. Ces expérimentations ont permis d'avoir un retour sur le jeu et donnent

quelques premiers éléments de réponses quant à la pertinence d'un tel jeu sérieux pour l'apprentissage des fondamentaux de la programmation.

# Conclusions et perspectives

Ce travail de recherche s'est donc attaché à étudier un jeu sérieux sur les fondamentaux de la programmation. Une étude préliminaire a consisté à positionner les jeux sérieux vis-à-vis des jeux vidéo et des simulateurs. L'objectif de cette étude était de bien comprendre les tenants et les aboutissants des jeux sérieux en vue d'en concevoir un, adapté à la pratique de la programmation. La raison d'une recherche sur ce sujet est motivée par « *la crise de l'informatique* » dans l'enseignement. Pour encourager les étudiants à revenir vers cette discipline, l'utilisation de nouvelle solutions pédagogiques doivent être envisagées et l'utilisation de jeux sérieux en est une. Pour tenter de motiver les étudiants, l'approche envisagée consiste à baser le jeu sérieux sur un type de jeu reconnu par les étudiants. Le choix s'est orienté vers les jeux de STR. Dans ce cadre, une étude approfondie de ce type de jeu a été conduite en vue d'identifier les règles et buts associés à chaque mode de jeu. Fort de ces premières études théoriques, une approche pragmatique a été envisagée à travers la conception, la réalisation, la mise en pratique et l'évaluation d'un jeu sérieux sur les fondamentaux de la programmation. Ce dernier est actuellement structuré autour de trois composantes principales : un moteur de jeu, l'API Prog&Play et un mode de jeu (campagne ou escarmouche).

La difficulté principale de conception d'un jeu sérieux porte sur l'intégration du message à véhiculer d'une manière cohérente avec l'univers du jeu sans en dénaturer la composante ludique. Le travail réalisé lors de la conception de la campagne de *Kernel Panic* constitue réellement l'intégration du savoir à enseigner dans l'univers du jeu. L'appréciation de la campagne par les étudiants lors des différentes expérimentations tend à valider la réalisation du jeu sérieux du point de vue du divertissement et de la cohérence des problèmes posés. Du point de vue de l'apprentissage, de nouvelles expérimentations sont requises avant de tirer des conclusions définitives.

Une première perspective de travail consiste donc à poursuivre les expérimentations en intégrant d'une manière plus conséquente le jeu dans les formations. En effet, en augmentant la présence du jeu dans le calendrier des enseignements, l'influence de ce dernier sur les étudiants devrait s'en trouver amplifiée. Il serait alors plus facile d'en observer les conséquences réelles.

Le jeu de STR a servi de cadre à cette étude, mais il serait possible d'envisager la réalisation d'un jeu sérieux pour l'apprentissage de la programmation avec d'autres familles de jeux supports. Ceci pourrait permettre de proposer plusieurs types de jeux sérieux de manière à ce que les étudiants puissent en choisir un en fonction de leurs préférences. À ce propos, les enquêtes réalisées auprès des étudiants, sur leur pratique du jeu vidéo, sont à poursuivre afin de conserver un regard sur l'évolution de leurs centres d'intérêts.

Une deuxième perspective de travail à plus long terme consiste donc à concevoir, réaliser et évaluer des jeux sérieux pour l'apprentissage de la programmation basés sur des types de jeux autres que les jeux de STR. Il serait alors intéressant de comparer l'amusement, l'apprentissage ou la facilité d'intégrer le savoir à enseigner pour chaque type de jeu.

Au cours du développement de la bibliothèque Prog&Play, la partie « Client » de l'API a été rendu compatible avec l'environnement de programmation Scratch. Ce portage offre trois avantages majeurs : premièrement le jeu bénéficie des atouts de la programmation à base de blocs (simplicité de manipulation, erreurs de syntaxe impossibles, etc.), deuxièmement, il profite des avantages d'un environnement de programmation (interaction avec le jeu et exécution des programmes simplifiés) et troisièmement, il est rendu accessible à un public plus jeune de lycéens et collégiens.

Cette combinaison entre Scratch et le jeu sérieux forme donc une troisième perspective de recherche pour étudier l'impact de ces deux technologies sur la motivation des étudiants. D'autre part, avec la réforme des lycées et l'intégration de l'algorithmique au programme de Mathématiques de seconde générale et technologique, Scratch est nommé comme l'un des outils utilisable en classe. Par conséquent, l'utilisation combinée de Scratch et du jeu sérieux ouvre la voie à de nouveaux contextes d'expérimentation sur un public plus jeune en cours d'orientation.

Du point de vue des expérimentations, l'effort a porté sur l'évaluation d'un jeu sérieux abordant les fondamentaux de la programmation. À ce titre, une campagne de sept missions a été conçue. Les expérimentations réalisées ont été menées avec des étudiants novices voire débutants en programmation et ont montré un réel intérêt de leur part pour ce type de pédagogie.

De nombreuses expérimentations peuvent encore être menées et constituent une quatrième perspective de recherche. Un premier exemple porte sur l'étude de l'approche par projet associée au mode escarmouche. Ce type de mise en œuvre peut laisser une part d'initiative plus importante aux étudiants et favoriser le travail en autonomie et en équipe. Un second exemple porte sur la réalisation de nouvelles campagnes pour aborder des concepts de programmation avancés tels que la programmation parallèle ou évènementielle, les concepts orientés objet (héritage, polymorphisme, etc.) ou encore les communications réseaux. De cette manière, le jeu sérieux pourrait être testé dans des niveaux académiques plus élevés. La confrontation du jeu sérieux avec des programmeurs confirmés peut laisser place à des études intéressantes. Dans tous les cas, l'évaluation de ces expérimentations doit solliciter une attention particulière afin d'identifier les composantes responsables des résultats obtenus (le type de jeu, *Kernel Panic*, le langage utilisé, l'interface de l'API Prog&Play, la mise en œuvre, le mode de jeu, etc.).

Enfin, tout au long de ces travaux et des rencontres effectuées autour de ce projet, un questionnement récurrent portait sur l'efficacité d'une telle approche basée sur un jeu sérieux. Du point de vue de la motivation, les retours des étudiants et des enseignants valident l'intérêt du jeu sérieux. En s'appuyant alors sur les travaux de Viau et du lien étroit entre motivation et performance, il est possible d'inférer une influence positive et indirecte du jeu sur les performances des étudiants. Toutefois, il convient de ne pas généraliser ces résultats, en effet, il n'existe pas à l'heure actuelle de méthode pédagogique efficace pour tous les étudiants quel que soit le contexte d'apprentissage. Le jeu sérieux est une solution parmi tant d'autres et l'expérience accumulée au cours des expérimentations permet de définir un cadre dans lequel l'utilisation de ce jeu sérieux est envisageable. Tout d'abord, le jeu sérieux semble trouver sa place en tant que complément aux formations classiques. Le soutien ou des ateliers de programmation sont particulièrement bien adaptés à son déploiement, car ils laissent le temps aux étudiants de concevoir et réaliser leurs solutions aux problèmes posés (missions ou projets). Ensuite, le choix d'utiliser le jeu doit être laissé aux étudiants. Par exemple, si un projet est envisagé au cours d'une formation, deux sujets pourront être proposés dont un sera basé sur le jeu sérieux. À ce propos, si la progression du marché du jeu vidéo continue sur sa lancée, le nombre d'étudiants potentiellement intéressé par ce type d'approche devrait s'en trouver également augmenté. Enfin, intégrer le jeu à une formation suppose une double adaptation : premièrement le jeu doit être ajusté au profil des étudiants via l'interface de programmation et deuxièmement le calendrier des enseignements de la formation devra être modifié pour laisser une place suffisante au jeu. Le respect de ces trois recommandations favorisera une mise en œuvre réussie du jeu sérieux comme élément de dynamisation des formations informatiques, de motivation et de facteur de réussite pour les étudiants.

# Bibliographie


[AAIC05] ACM, AIS, and IEEE-CS. *Computing Curricula 2005, The Overview Report*. ACM Press. and IEEE Computer Society Press., New York, 2005.

[Abt70] C. Abt. *Serious Games*. University Press of America, 1970.

[AIC01] ACM and IEEE-CS. *Computing Curricula 2001, Computer Science*. ACM Press. and IEEE Computer Society Press., New York, 2001.

[AIC08] ACM and IEEE-CS. *Computer Science Curriculum 2008 : An Interim Revision of CS2001*. ACM Press. and IEEE Computer Society Press., New York, 2008.

[Alv07] J. Alvarez. *Du jeu vidéo au serious game, approches culturelle, pragmatique et formelle*. PhD thesis, Université de Toulouse, 2007.

[Ass09] Entertainment Software Association. Essential facts about the computer and video game industry. `http://www.theesa.com/facts/pdfs/ESA_EF_2009.pdf` accédé le 28 juin 2010, 2009.

[Ban86] A. Bandura. *Social Foundations of Thought and Action : A Social Cognitive Theory*. Englewood Cliffs (N. J.) : Prentice-Hall, 1986.

[Ban97] A. Bandura. *Self-efficacy : The exercise of control*. New York : Worth Publishers, 1997.

[BF05] M. Buro and T. Furtak. On the development of a free rts game engine, March 2005. GameOn'NA Conference.

[Bla05] S. Blackman. Serious games... and less ! *SIGGRAPH Comput. Graph.*, 39(1) :12–16, 2005.

[Blu06] R. Blunt. Game-based learning works : Teaching business courses with video games. In *Serious Games Summit D.C.*, Washington, DC, USA, oct 2006.

[BT01] P. Bettner and M. Terrano. 1500 archers on a 28.8 : Network programming an age of empires and beyond. In *GDC*, mar 2001.

[Bul09] Buletin officiel n°30. Mathématiques, classe de seconde. `http://media.education.gouv.fr/file/30/52/3/programme_mathematiques_seconde_65523.pdf` accédé le 21 juillet 2010, publié le 23 Juillet 2009.

[Bur02] M. Buro. ORTS : A hack-free RTS game environment. In *Computers and Games*, pages 280–291, 2002.

[CB98] A. Cockburn and A. Bryant. Cleogo : Collaborative and multi-metaphor programming for kids. In *APCHI '98 : Proceedings of the Third Asian Pacific Computer and Human Interaction*, page 189, Washington, DC, USA, 1998. IEEE Computer Society.

[CC07] W.-K. Chen and Y. C. Cheng. Teaching object-oriented programming laboratory with computer game programming. *IEEE Transactions on Education*, 50(3) :197–203, aug 2007.

[CCD+03] P. Cobb, J. Confrey, A. DiSessa, R. Lehrer, and L. Schauble. Design experiments in educational research. *Educational Researcher*, 32(1) :9–13, January 2003.

[CCMT08] T. L. Crenshaw, E. W. Chambers, H. Metcalf, and U. Thakkar. A case study of retention practices at the university of illinois at urbana-champaign. *39th ACM Technical Symposium on Computer Science Education*, 40(1) :412–416, 2008.

[CDP03] S. Cooper, W. Dann, and R. Pausch. Teaching objects-first in introductory computer science. In *SIGCSE '03 : Proceedings of the 34th SIGCSE technical symposium on Computer science education*, pages 191–195, New York, NY, USA, 2003. ACM.

[CHHY04] D. J. Cook, L. B. Holder, M. Huber, and R. Yerraballi. Enhancing computer science education with a wireless intelligent simulation environment. *Journal of Computing in Higher Education*, 16(1), 2004.

[Cra84] C. Crawford. *The Art of Computer Game Design*. Osborne/McGraw-Hill, Berkeley, CA, USA, 1984.

[DZ03] T. N. B. Duong and S. Zhou. A dynamic load sharing algorithm for massively multiplayer online games. In *The 11th IEEE International Conference on Networks*, pages 131–136, 2003.

[Edu09] EduSCOL, Ministère de l'éducation nationale. Ressources pour la classe de seconde - algorithmique. http://media.eduscol.education.fr/file/Programmes/17/8/Doc_ress_algo_v25_109178.pdf accédé le 21 juillet 2010, publié en Juin 2009.

[FMT06] P. Fuchs, G. Moreau, and J. Tisseau. *Traité de la réalité virtuelle, Tome III.*, chapter 1. Presses de l'École des Mines de Paris, 2006.

[FPC+04] S. Fincher, M. Petre, M. Clancy, T. Clear, M. Guzdial, C. D. Hundhausen, W. M. McCracken, R. S. Rist, and J. T. Stasko. *Computer Science Education Research*, chapter 1, pages 1–8. Taylor & Francis The Netherlands, Lisse, 2004.

[Gin04] D. Ginat. On novice loop boundaries and range conceptions. *Computer Science Education*, 14(3) :165–181, 2004.

[GKH07] F. L. Greitzer, O. A. Kuchar, and K. Huston. Cognitive science implications for enhancing training effectiveness in a serious gaming context. *J. Educ. Resour. Comput.*, 7(3) :2, 2007.

[GS08] P. Gestwicki and F.-S. Sun. Teaching design patterns through computer game development. *ACM Journal on Educational Resources in Computing*, 8(1) :1–22, 2008.

[Har04] K. Hartness. Robocode : using games to teach artificial intelligence. *J. Comput. Small Coll.*, 19(4) :287–291, 2004.

[JVM05] W. Lewis Johnson, H. Vilhjalmsson, and S. Marsella. Serious games for language learning : How much game, how much AI ? In *12th International Conference on Artificial Intelligence in Education (AIED)*, Amsterdam, The Netherlands, july 2005.

[Kel06] C. Kelleher. Alice and the sims : the story from the alice side of the fence. In *The Annual Serious Games Summit DC Washington*, DC, USA, October 30–31, 2006.

[KLXH04] B. Knutsson, H. Lu, W. Xu, and B. Hopkins. Peer-to-peer support for massively multiplayer games, March 2004. IEEE Infocom.

[KPK07] C. Kelleher, R. Pausch, and S. Kiesler. Storytelling alice motivates middle school girls to learn computer programming. In *CHI '07 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1455–1464, New York, NY, USA, 2007. ACM.

[KY05] E. Klopfer and S. Yoon. Developing games and simulations for today and tomorrow's tech savvy youth. *Tech Trends*, 49(3) :33–41, 2005.

[LE07] S. Leutenegger and J. Edgington. A games first approach to teaching introductory programming. *SIGCSE '07 : Proceedings of the 38th SIGCSE technical symposium on Computer science education*, 39(1) :115–118, mar 2007.

[MBK+04] J. Maloney, L. Burd, Y. Kafai, N. Rusk, B. Silverman, and M. Resnick. Scratch : A sneak preview. In *2nd International Conference on Creating Connecting, and Collaborating through Computing*, pages 104–109, Keihanna-Plaza, Kyoto, Japan, jan 2004.

[Min05] Ministère de l'éducation nationale, de l'enseignement supérieur et de la recherche. Programme Pédagogique National du DUT « Informatique ». `http://media.enseignementsup-recherche.gouv.fr/file/77/6/776.pdf` accédé le 28 juin 2010, 2005.

[MTJ09] M. Muratet, P. Torguet, and J.-P. Jessel. Learning programming with an rts-based serious game. In Otto Petrovic and Anthony Brand, editors, *Serious Games on the Move*, pages 181–192. Springer Vienna, 2009.

[MTJV08a] M. Muratet, P. Torguet, J.-P. Jessel, and F. Viallet. Apprentissage de la programmation à l'aide d'un jeu sérieux. In *Ludovia*, page (en ligne), `http://www.ludovia.org`, août 2008. Ludovia.

[MTJV08b] M. Muratet, P. Torguet, J.-P. Jessel, and F. Viallet. Vers un jeu sérieux pour enseigner la programmation. In *Association Française de Réalité Virtuelle, Augmentée, Mixte et d'Interaction 3D (AFRV)*, page (en ligne). AFRV, octobre 2008.

[MTJV09] M. Muratet, P. Torguet, J.-P. Jessel, and F. Viallet. Towards a serious game to help students learn computer programming. *International Journal of Computer Games Technology*, 2009 :(en ligne), 2009.

[MTVJ10a] M. Muratet, P. Torguet, F. Viallet, and J.-P. Jessel. Experimental feedback on prog&play : a serious game for programming practice. *Computer Graphics Forum*, 2010.

[MTVJ10b] M. Muratet, P. Torguet, F. Viallet, and J.-P. Jessel. Experimental feedback on prog&play, a serious game for programming practice (educational paper). In *Eurographics (EG)*, page (média électronique), `http://www.eg.org/`, 2010. Eurographics.

[MVTJ09] M. Muratet, F. Viallet, P. Torguet, and J.-P. Jessel. Une ingénierie pour jeux sérieux. In *EIAH - Atelier Jeux Sérieux : conception et usages*, pages 53–63. Sébastien George et Éric Sanchez, juin 2009.

[NWFR06] V. Narayanasamy, K. W. Wong, C. C. Fung, and S. Rai. Distinguishing games and simulation games from simulators. *Comput. Entertain.*, 4(2) :9, 2006.

[Pia94] J. Piaget. *La formation du symbole chez l'enfant : imitation, jeu et rêve, image et représentation*. Paris, Delachaux et Niestlé, 1994.

[PMB93] P. R. Pintrich, R. W. Marx, and R. A. Boyle. Beyond cold conceptual change : the role of motivational beliefs and classroom contextual factors in the process of contextual change. *Educational Research*, 630(2) :167–199, 1993.

[PS96] P. R. Pintrich and D. H. Schunk. *Motivation in Education : theory, research and applications*. Englewood Cliffs : Prentice Hall, 1996.

[SBE83] E. Soloway, J. Bonar, and K. Ehrlich. Cognitive strategies and looping constructs : An empirical study. *Communications of the ACM*, 26(11) :853–860, 1983.

[SC05] L. Smith and J. Cordova. Weighted primary trait analysis for computer program evaluation. *J. Comput. Small Coll.*, 20(6) :14–19, 2005.

[SMK06] O. Seppälä, L. Malmi, and A. Korhonen. Observations on student misconceptions - a case study of the build heap algorithm. *Computer Science Education*, 16(3) :241–255, 2006.

[SR03] A. C. Siang and K. R. Radha. Theories of learning : a computer game perspective. In *Fifth International Symposium on Multimedia Software Engineering, 2003. Proceedings*, pages 239–245, dec 2003.

[Sta07] S. Stamm. Mixed nuts : atypical classroom techniques for computer science courses. *Crossroads The ACM Student Magazine*, 10(4), 2007.

[Ste04] C. A. Steinkuehler. Learning in massively multiplayer online games. In *ICLS '04 : Proceedings of the 6th international conference on Learning sciences*, pages 521–528. International Society of the Learning Sciences, 2004.

[SW06] D. E. Stevenson and P. J. Wagner. Developing real-world programming assignments for cs1. In *ITICSE '06 : Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education*, pages 158–162, Bologna, Italy, jun 2006.

[Via97] R. Viau. *La motivation en contexte scolaire*. Bruxelles : De Boeck, 1997.

[VMD+10] V. Viallet, M. Muratet, M. David, J.-L. Bach, P. Torguet, and O. Catteaun. Mutualisation en iut d'un jeu sérieux pour l'apprentissage de la programmation. In *CIUEN - Colloque International de l'Université à l'Ére du Numérique*. (à paraître), juin 2010.

[Wei92] B. Weiner, editor. *Human Motivation*. Newbury Park (CA) : Sage, 1992.

[Wol02] M. J. P. Wolf, editor. *The Medium of the Video Game*. University of Texas Press, 2002. Foreword By-Baer, Ralph H.

[Zim86] B. J. Zimmerman. Becoming a self-regulated learner : Which are the key subprocesses ? *Contemporary Educational Psychology*, 11(4) :307–313, 1986.

[Zyd05] M. Zyda. From visual simulation to virtual reality to games. *Computer*, 38(9) :25–32, 2005.

# Troisième partie

# Annexes

# A

# Détail des interfaces de l'API Prog&Play en langage C

## A.1 Interface « Fournisseur »

### A.1.1 Types de données

```
#define NB_COALITIONS 3

/* Definit les coalitions possibles */
typedef enum {
  MY_COALITION,
  ALLY_COALITION,
  ENEMY_COALITION
} PP_Coalition;

/* Definit une position */
typedef struct {
  float x;
  float y;
} PP_Pos;

/* Definit une liste de positions */
typedef struct {
        int size;
        PP_Pos *pos;
} PP_Positions;

/* Identifiant d'une unite */
typedef int PP_Unit;
```

```
/* Contenu d'une unite */
typedef struct {
        PP_Unit id;
        PP_Coalition coalition;
        int type;
        PP_Pos pos;
        float health;
        float maxHealth;
        int group;
        int nbCommandQueue;
        int *commandQueue;
} PP_ShortUnit;

/* Definit une liste d'unites */
typedef struct{
        int size;
        PP_ShortUnit *unit;
} PP_Units;

/* Contenu d'une commande */
typedef struct{
        PP_Unit unitId;
        int group; /* groupe d'affectation */
        /* Type de commande
         * -1 : commande invalide
         * 0 : la cible de la commande est une position
         * 1 : la cible de la commande est une unite
         * 2 : la commande n'a pas de cible */
        int commandType;
        int commandCode; /* code de la commande */
        int nbParam; /* nombre de paramettres */
        float *param; /* liste des paramettres */
} PP_ShortCommand;

/* Definit une liste de commandes */
typedef struct{
        int size;
        PP_ShortCommand *command;
} PP_Commands;

/* Identifiant d'une ressource */
typedef int PP_Resource;

/* Definit une liste de ressources */
typedef struct{
        int size;
        PP_Resource *resource;
} PP_Resources;
```

### A.1.2 Entêtes des fonctions

int **PP_Init**(void)

Initialise l'API Prog&Play. Cette fonction doit être appelée avant l'utilisation de toute autre fonction de l'API.
**Retourne** 0 en cas de succès et -1 sinon.

int **PP_Quit**(void)

Ferme et nettoie l'API Prog&Play. Après appel de cette fonction, aucune autre fonction de l'API ne doit être appelée à l'exception de PP_Init.
**Retourne** 0 en cas de succès et -1 sinon.

int **PP_NeadUpdate**(void)

**Retourne** une valeur positive si une mise à jour est demandée et 0 sinon. -1 est retourné en cas d'erreur.

int **PP_Update**(const PP_Units **units*, const int **gameOver*, const PP_Pos **mapSize*, const PP_Pos **startPos*, const PP_Positions **specialAreas*, const PP_Resources **resources*)

***units*** : nouvel ensemble des unités. Si NULL, l'ancien ensemble est conservé.
***gameOver*** : nouvel état de fin de jeu. Si NULL, l'ancien état est conservé. Si *gameOver != 0, la partie est terminée. Si *gameOver == 0, la partie est en cours de déroulement.
***mapSize*** : nouvelle taille de la carte du jeu. Si NULL, l'ancienne taille est conservée.
***startPos*** : nouvelle position de départ du joueur sur la carte. Si NULL, l'ancienne position est conservée.
***specialAreas*** : nouvel ensemble des positions des zones spéciales. Si NULL, l'ancien ensemble est conservé.
***resources*** : nouvel ensemble du niveau des ressources. Si NULL, l'ancien ensemble est conservé.

Met à jour l'état du jeu pour le « client ». Tous les paramètres en entrée doivent être des pointeurs sur des données consistantes ou NULL. Toutes ces données sont recopiées, il incombe à l'utilisateur de libérer la mémoire associée à ces données.
**Retourne** 0 en cas de succès et -1 sinon.

PP_Commands* **PP_GetPendingCommands**(void)

Fournit les commandes définies par le « client ». Utiliser PP_FreePendingCommand pour libérer les commandes récupérées à travers cette fonction.
**Retourne** les commandes définies par le « client ». NULL est retourné en cas d'erreur.

void **PP_FreePendingCommands**(PP_Commands **commands*)

***commands*** : ensemble des commandes en attente à libérer.

Libère la zone mémoire pointée par *commands*. Ne pas référencer cette zone mémoire après l'appel de cette fonction.
**Retourne** 0 en cas de succès et -1 sinon.

## A.2 Interface « Client »

### A.2.1 Types de données

```
#define NB_COALITIONS 3

/* Definit les coalitions possibles */
typedef enum {
  MY_COALITION,
  ALLY_COALITION,
  ENEMY_COALITION
} PP_Coalition;

/* Definit une position */
typedef struct {
  float x;
  float y;
} PP_Pos;

/* Identifiant d'une unite */
typedef int PP_Unit;

/* Identifiant d'une ressource */
typedef int PP_Resource;
```

### A.2.2 Entêtes des fonctions

int **PP_Open**(void)

Ouvre l'API Prog&Play. Cette fonction doit être appelée avant l'utilisation de toute autre fonction de l'API.
**Retourne** 0 en cas de succès et -1 sinon.

int **PP_Close**(void)

Ferme l'API Prog&Play. Après appel de cette fonction, aucune autre fonction de l'API ne doit être appelée à l'exception de PP_Open.
**Retourne** 0 en cas de succès et -1 sinon.

int **PP_Refresh**(void)

Demande au jeu de mettre à jour les données. Cette procédure est bloquante jusqu'à ce que le jeu ait terminé sa mise à jour. **Retourne** 0 en cas de succès et -1 sinon.

int **PP_IsGameOver**(void)

**Retourne** une valeur positive si la partie est terminée et 0 sinon. -1 est retourné en cas d'erreur.

PP_Pos **PP_GetMapSize**(void)

**Retourne** la taille de la carte sur laquelle se déroule la partie en cas de succès. La position {-1.0, -1.0} est retournée en cas d'erreur.

PP_Pos **PP_GetStartPosition**(void)

**Retourne** la position de départ du joueur sur la carte en cas de succès. La position {-1.0, -1.0} est retournée en cas d'erreur.

int **PP_GetNumSpecialAreas**(void)

**Retourne** le nombre de zones spéciales en cas de succès et -1 sinon.

PP_Pos **PP_GetSpecialAreaPosition**(int *num*)

***num*** : identifiant de la zone spéciale. Il doit être inclus dans l'intervalle $[0; n[$ où $n$ est le nombre de zones spéciales (voir PP_GetNumSpecialAreas).

**Retourne** la position de la zone spéciale en cas de succès. La position {-1.0, -1.0} est retournée en cas d'erreur.

int **PP_GetResource**(PP_Resource *id*)

***id*** : identifiant de la ressource dont le niveau de réserve doit être connu.

**Retourne** le niveau de réserve courant en cas de succès et -1 sinon.

int **PP_GetNumUnits**(PP_Coalition *c*)

***c*** : coalition à consulter.

**Retourne** le nombre d'unités (visibles par le joueur) de cette coalition en cas de succès et -1 sinon.

PP_Unit **PP_GetUnitAt**(PP_Coalition *c*, int *index*)

***c*** : coalition à consulter.
***index*** : i$^{\text{ème}}$ unité visible de la coalition *c*. *index* doit être inclus dans l'intervalle $[0; n[$ où $n$ est le nombre d'unités visibles de la coalition *c* (voir PP_GetNumUnits).

**Retourne** la i$^{\text{ème}}$ unité visible de la coalition *c* en cas de succès et -1 sinon.

PP_Coalition **PP_Unit_GetCoalition**(PP_Unit *unit*)

***unit*** : unité à consulter.

**Retourne** la coalition de l'unité spécifiée en cas de succès et -1 sinon.

int **PP_Unit_GetType**(PP_Unit *unit*)

***unit*** : unité à consulter.

**Retourne** le type de l'unité spécifiée en cas de succès et -1 sinon.

PP_Pos **PP_Unit_GetPosition**(PP_Unit *unit*)

***unit*** : unité à consulter.

**Retourne** la position de l'unité spécifiée en cas de succès. La position {-1.0, -1.0} est retournée en cas d'erreur.

float **PP_Unit_GetHealth**(PP_Unit *unit*)

***unit*** : unité à consulter.

**Retourne** le capital santé de l'unité spécifiée en cas de succès et -1.0 sinon.

float **PP_Unit_GetMaxHealth**(PP_Unit *unit*)

***unit*** : unité à consulter.

**Retourne** le capital santé maximum de l'unité spécifiée en cas de succès et -1.0 sinon.

int **PP_Unit_GetGroup**(PP_Unit *unit*)

***unit*** : unité à consulter.

**Retourne** le numéro de groupe de l'unité spécifiée en cas de succès. -2 est retourné si l'unité spécifiée n'est affectée à aucun groupe. -1.0 est retourné en cas d'erreur.

int **PP_Unit_SetGroup**(PP_Unit *unit*, int *group*)

***unit*** : unité à commander. Seules les unités contrôlées par le joueur peuvent recevoir cet ordre.
***group*** : groupe d'affectation (>= 0). Si *group* == -1 alors l'unité spécifiée est retirée de son groupe.

Affecte l'unité *unit* dans le groupe *group*.
**Retourne** 0 en cas de succès et -1 sinon.

int **PP_Unit_GetNumPendingCommands**(PP_Unit *unit*)

***unit*** : unité à consulter. Seules les unités contrôlées par le joueur peuvent fournir cette information.

**Retourne** le nombre de commandes empilées pour cette unité en cas de succès et -1 sinon.

int **PP_Unit_GetPendingCommandAt**(PP_Unit *unit*, int *index*)

***unit*** : unité à consulter. Seules les unités contrôlées par le joueur peuvent fournir cette information.
***index*** : i$^{\text{ème}}$ commande en attente de l'unité spécifiée. *index* doit être inclus dans l'intervalle $[0; n[$ où $n$ est le nombre de commandes empilées pour l'unité *unit* (voir PP_Unit_GetNumPendingCommands).

**Retourne** la i$^{\text{ème}}$ commande de l'unité *unit* en cas de succès et -1 sinon.

int **PP_Unit_ActionOnUnit**(PP_Unit *unit*, int *action*, PP_Unit *target*)

***unit*** : unité à commander. Seules les unités contrôlées par le joueur peuvent recevoir cet ordre.
***action*** : action à réaliser.
***target*** : unité cible.

Ordonne à l'unité *unit* de réaliser l'action spécifiée sur l'unité *target*.
**Retourne** 0 en cas de succès et -1 sinon.

int **PP_Unit_ActionOnPosition**(PP_Unit *unit*, int *action*, PP_Pos *pos*)

***unit*** : unité à commander. Seules les unités contrôlées par le joueur peuvent recevoir cet ordre.
***action*** : action à réaliser.
***target*** : position cible.

Ordonne à l'unité *unit* de réaliser l'action spécifiée à la position *target*.
**Retourne** 0 en cas de succès et -1 sinon.

int **PP_Unit_UntargetedAction**(PP_Unit *unit*, int *action*, float *param*)

***unit*** : unité à commander. Seules les unités contrôlées par le joueur peuvent recevoir cet ordre.
***action*** : action à réaliser.
***param*** : paramètre de l'action. Si aucun paramètre n'est requis pour l'action spécifiée, indiquez la valeur -1.0.

Ordonne à l'unité *unit* de réaliser l'action spécifiée avec le paramètre *param*.
**Retourne** 0 en cas de succès et -1 sinon.

## A.3 Interface de gestion des erreurs

char* **PP_GetError**(void)

Fournit la dernière erreur générée par l'API Prog&Play.
**Retourne** la dernière erreur définie comme une chaîne de caractères terminée par NULL. Cette chaîne est allouée statiquement et n'a pas à être libérée par l'utilisateur.

void **PP_ClearError**(void)

Supprime toutes les informations de la dernière erreur générée. Utile si l'erreur a été traitée.

# B

# Implémentation de la liste de constantes de *Kernel Panic 3.8* en langage C

```
#ifndef CONSTANT_LIST_KP38
#define CONSTANT_LIST_KP38
/*
 * Liste des constantes pour les unites des "Systemes" de Kernel Panic 3.8
 */

/*************************
 * identifiant des unites *
 *************************/
#define ASSEMBLER           2
#define BADBLOCK            3
#define BIT                 4
#define BYTE                7
#define KERNEL              24
#define LOGIC_BOMB          25
#define POINTER             37
#define SIGNAL              40
#define SOCKET              41
#define TERMINAL            42

/*********************************************************************
 * Ordres disponibles pour : ASSEMBLER, BIT, BYTE, KERNEL, LOGIC_BOMB, *
 * POINTER et SOCKET                                                  *
 *********************************************************************/
#define STOP                0     /* attend 0 parametre */
#define WAIT                5     /* attend 0 parametre */
#define FIRE_STATE          45    /* attend 1 parametre :
                                        0.0 => Cesser le feu
                                        1.0 => Riposte
```

```
                                    2.0 => Feu a volonte */
#define SELF_DESTRUCTION  65    /* attend 0 parametre */
#define REPEAT            115   /* attend 1 parametre :
                                    0.0 => Repetition desactivee
                                    1.0 => Repetition activee */
/*****************************************************************
 * Ordres disponibles pour : ASSEMBLER, BIT, BYTE, KERNEL, POINTER et *
 * SOCKET                                                        *
 *****************************************************************/
#define MOVE              10    /* attend 1 parametre :
                                    une position ou une unite */
#define PATROL            15    /* attend 1 parametre :
                                    une position ou une unite */
#define FIGHT             16    /* attend 1 parametre :
                                    une position ou une unite */
#define GUARD             25    /* attend 1 parametre :
                                    une position ou une unite */
#define MOVE_STATE        50    /* attend 1 parametre :
                                    0.0 => Tenir position
                                    1.0 => Manoeuvrer
                                    2.0 => Vadrouiller */
/*****************************************************************
 * Ordres disponibles pour : BIT, BYTE, KERNEL, LOGIC_BOMB, POINTER *
 * et SOCKET                                                     *
 *****************************************************************/
#define ATTACK            20    /* attend 1 parametre :
                                    une position ou une unite */
/************************************
 * Ordres disponibles pour : ASSEMBLER *
 ************************************/
#define REPAIR            40    /* attend 1 parametre :
                                    une position ou une unite */
#define RECLAIM           90    /* attend 1 parametre :
                                    une position ou une unite */
#define RESTORE           110   /* attend 1 parametre :
                                    une position ou une unite */
#define BUILD_BADBLOCK    -3    /* attend 1 parametre :
                                    une position ou une unite */
#define BUILD_LOGIC_BOMB  -25   /* attend 1 parametre :
                                    une position ou une unite */
#define BUILD_SOCKET      -41   /* attend 1 parametre :
                                    une position ou une unite */
#define BUILD_TERMINAL    -42   /* attend 1 parametre :
                                    une position ou une unite */
#define DEBUG             -34   /* attend 1 parametre :
                                    une position ou une unite */
/**********************************
 * Ordres disponibles pour : KERNEL *
 **********************************/
#define BUILD_ASSEMBLER   -2    /* attend 1 parametre :
```

```
                                        une position ou une unite */
#define BUILD_BYTE          -7      /* attend 1 parametre :
                                        une position ou une unite */
#define BUILD_POINTER       -37     /* attend 1 parametre :
                                        une position ou une unite */
/*******************************************
 * Ordres disponibles pour : KERNEL et SOCKET *
 *******************************************/
#define BUILD_BIT           -4      /* attend 1 parametre :
                                        une position ou une unite */
/*********************************
 * Ordres disponibles pour : BYTE *
 *********************************/
#define LAUNCH_MINE         33395 /* attend 0 parametre */
/************************************
 * Ordres disponibles pour : POINTER *
 ************************************/
#define NX_FLAG             33389 /* attend 1 parametre :
                                        une position ou une unite */
/************************************
 * Ordres disponibles pour : TERMINAL *
 ************************************/
#define SIGTERM             35126 /* attend 1 parametre :
                                        une position ou une unite */

/*****************************/
/* Identifiant des ressources */
/*****************************/
#define METAL  0
#define ENERGY 1

#endif
```

# C

# Solution en langage C des six premières missions de la campagne

## C.1 Mission 1

```
#include "PP_Client.h"
#include "constantList_KP3.8.h"

/* Solution Mission 1 */
int main (void){
  PP_Pos p; /* Position a atteindre */
  PP_Unit u; /* Unite courante */

  /* Definition de la position cible */
  p.x = 1983.0;
  p.y = 1279.0;
  PP_Open(); /* Ouverture de l'API Prog&Play */
  PP_Refresh(); /* Demande d'une mise a jour des donnees */
  u = PP_GetUnitAt(MY_COALITION, 0); /* Recuperation de la premiere unite */
  /* Ordonner a cette unite de se deplacer a la position cible */
  PP_Unit_ActionOnPosition(u, MOVE, p);
  PP_Close(); /* Fermeture de l'API Prog&Play */
  return 0;
}
```

## C.2 Mission 2

```
#include "PP_Client.h"
#include "constantList_KP3.8.h"
#include <math.h>
```

```
/* Solution Mission 2 */
int main (void){
  PP_Unit u; /* Unite courante */
  double r; /* angle en radian */
  PP_Pos posArrivee; /* Position a atteindre */

  PP_Open(); /* Ouverture de l'API Prog&Play */
  PP_Refresh(); /* Demande d'une mise a jour des donnees */
  u = PP_GetUnitAt(MY_COALITION, 0); /* Recuperation de la premiere unite */
  /* Recuperation des coordonnees de l'unite */
  PP_Pos posDepart = PP_Unit_GetPosition(u);
  /* Calcul de l'angle en radian (repere trigonometrique) */
  r = 3.14159265*(119+90)/180;
  /* Calcul de la coordonnee d'arrivee */
  /* Attention : l'origine du repere est situe en haut a gauche de la carte */
  posArrivee.x = posDepart.x + cos (r) * 1060;
  posArrivee.y = posDepart.y - sin (r) * 1060;
  /* Ordonner a cette unite de se deplacer a la position cible */
  PP_Unit_ActionOnPosition(u, MOVE, posArrivee);
  PP_Close(); /* Fermeture de l'API Prog&Play */
  return 0;
}
```

## C.3 Mission 3

```
#include "PP_Client.h"
#include "constantList_KP3.8.h"

/* Solution Mission 3 */
int main (void){
  PP_Pos bits; /* Point de ralliement des BITS */
  PP_Pos bytes; /* Point de ralliement des OCTETS */
  PP_Unit u; /* Unite courante */

  /* Definition des points de ralliement */
  bits.x = 1400.0;
  bits.y = 1371.0;
  bytes.x = 479.0;
  bytes.y = 1825.0;
  PP_Open(); /* Ouverture de l'API Prog&Play */
  PP_Refresh(); /* Demande d'une mise a jour des donnees */
  u = PP_GetUnitAt(MY_COALITION, 0); /* Recuperation de la premiere unite */
  /* Evaluation du type de l'unite */
  if (PP_Unit_GetType(u) == BIT){
    /* Ordonner a cette unite de rejoindre le point de ralliement des BITS */
    PP_Unit_ActionOnPosition(u, MOVE, bits);
    /* Passer a l'unite suivante */
    u = PP_GetUnitAt(MY_COALITION, 1);
```

```
    /* Ordonner a cette unite de rejoindre le point de ralliement des OCTETS */
    PP_Unit_ActionOnPosition(u, MOVE, bytes);
  }
  else {
    /* Ordonner a cette unite de rejoindre le point de ralliement des OCTETS */
    PP_Unit_ActionOnPosition(u, MOVE, bytes);
    /* Passer a l'unite suivante */
    u = PP_GetUnitAt(MY_COALITION, 1);
    /* Ordonner a cette unite de rejoindre le point de ralliement des BITS */
    PP_Unit_ActionOnPosition(u, MOVE, bits);
  }
  PP_Close(); /* Fermeture de l'API Prog&Play */
  return 0;
}
```

# C.4 Mission 4

```
#include "PP_Client.h"
#include "constantList_KP3.8.h"

/* Solution Mission 4 */
int main (void){
  int i; /* Compteur de boucle */
  PP_Unit u; /* Unite courante */
  PP_Pos pos; /* Position a atteindre */

  /* Definition du point de ralliement */
  pos.x = 256.0;
  pos.y = 1024.0;
  PP_Open(); /* Ouverture de l'API Prog&Play */
  PP_Refresh(); /* Demande d'une mise a jour des donnees */
  /* Parcourir toutes les unites */
  i = 0;
  while (i < PP_GetNumUnits(MY_COALITION)){
    /* Recuperation de l'unite courante */
    u = PP_GetUnitAt(MY_COALITION, i);
    /* Ordonner a cette unite de rejoindre le point de ralliement */
    PP_Unit_ActionOnPosition(u, MOVE, pos);
    i++;
  }
  PP_Close(); /* Fermeture de l'API Prog&Play */
  return 0;
}
```

# C.5 Mission 5

```
#include "PP_Client.h"
```

```
#include "constantList_KP3.8.h"

/* Solution Mission 5 */
int main (void){
  int i; /* Compteur de boucle */
  PP_Unit u; /* Unite courante */
  PP_Pos pos; /* Position a atteindre */

  /* Definition du point de ralliement */
  pos.x = 256.0;
  pos.y = 811.0;
  PP_Open(); /* Ouverture de l'API Prog&Play */
  PP_Refresh(); /* Demande d'une mise a jour des donnees */
  /* Parcourir toutes les unites */
  i = 0;
  while (i < PP_GetNumUnits(MY_COALITION)){
    /* Recuperation de l'unite courante */
    u = PP_GetUnitAt(MY_COALITION, i);
    /* Evaluation du type de l'unite */
    if (PP_Unit_GetType(u) == ASSEMBLER){
      /* Ordonner a l'ASSEMBLEUR de rejoindre le point de ralliement */
      PP_Unit_ActionOnPosition(u, MOVE, pos);
      i = PP_GetNumUnits(MY_COALITION); /* Sortir de la boucle */
    }
    else
      i++;
  }
  PP_Close(); /* Fermeture de l'API Prog&Play */
  return 0;
}
```

# C.6 Mission 6

```
#include "PP_Client.h"
#include "constantList_KP3.8.h"

/* Solution Mission 6 */
int main (void){
  int i; /* Compteur de boucle */
  PP_Unit u, assembleur; /* Unites de traitement */

  PP_Open(); /* Ouverture de l'API Prog&Play */
  PP_Refresh(); /* Demande d'une mise a jour des donnees */
  /* Recherche de l'assembleur */
  i = 0;
  u = PP_GetUnitAt(MY_COALITION, i);
  while (i < PP_GetNumUnits(MY_COALITION) && PP_Unit_GetType(u) != ASSEMBLER){
    /* Passer a l'unite suivante */
    i = i + 1;
```

```
    u = PP_GetUnitAt(MY_COALITION, i);
  }
  /* Debuter la reparation */
  if (PP_Unit_GetType(u) == ASSEMBLER){
    assembleur = u;
    /* Parcourir toutes les unites */
    i = 0;
    while (i < PP_GetNumUnits(MY_COALITION)){
      /* Recuperation de l'unite courante */
      u = PP_GetUnitAt(MY_COALITION, i);
      /* Verifier si reparation necessaire */
      if (PP_Unit_GetHealth(u) < PP_Unit_GetMaxHealth(u)){
        /* Ordonner a l'ASSEMBLEUR de commencer la reparation de l'unite courante */
        PP_Unit_ActionOnUnit(assembleur, REPAIR, u);
        /* Attente de fin de reparation */
        while (PP_Unit_GetHealth(u) < PP_Unit_GetMaxHealth(u)){
          PP_Refresh();
        }
      }
      i = i + 1;
    }
  }
  PP_Close(); /* Fermeture de l'API Prog&Play */
  return 0;
}
```

# D

# Réalisation d'une interface algorithmique pour le langage C : « PP_ALGO.h »

```
#ifndef PP_ALGO
#define PP_ALGO

#include "PP_Client.h"
#include "constantList_KP3.8.h"

/*************************************/
/* Definition des mots clefs du langage */
/*************************************/
#define Debut int main (){
#define Fin }
#define Si if (
#define Alors ){
#define Sinon } else {
#define FinSi }
#define TantQue while (
#define Faire ){
#define FinTantQue }
#define Et &&
#define Ou ||
#define Non !
#define VRAI 1
#define FAUX 0

/*************************************************************/
/* Definition des operateurs d'interaction avec Kernel Panic */
/*************************************************************/
/* Pour gerer l'unite courante */
PP_Unit current_unit;
```

```
int current_id = 0;

/* Ouvre la connexion avec le jeu.
   Cet operateur doit etre appele avant tout autre operateur */
#define OUVRIR_JEU PP_Open(); PP_Refresh ()
/* Ferme la connexion avec le jeu.
   Cet operateur doit imperativement etre le dernier operateur a etre appele
   avant la fin du programme */
#define FERMER_JEU PP_Close ()
/* Initialise l'unite courante a la premiere unite controlee par le joueur */
#define PREMIERE_UNITE current_id = 0;\
                       current_unit = PP_GetUnitAt(MY_COALITION, current_id)
/* Fait passer l'unite courante a l'unite suivante controlee par le joueur */
#define UNITE_SUIVANTE current_id ++;\
                       current_unit = PP_GetUnitAt(MY_COALITION, current_id)
/* Fournit la derniere unite controlable par le joueur */
#define DERNIERE_UNITE PP_GetUnitAt(MY_COALITION, PP_GetNumUnits (MY_COALITION)-1)
/* Indique VRAI si l'unite courante est un BIT */
#define EST_UN_BIT (PP_Unit_GetType (current_unit) == BIT)
/* Indique VRAI si l'unite courante est un BYTE */
#define EST_UN_BYTE (PP_Unit_GetType (current_unit) == BYTE)
/* Indique VRAI si l'unite courante est un ASSEMBLEUR */
#define EST_UN_ASSEMBLEUR (PP_Unit_GetType (current_unit) == ASSEMBLER)
/* Donne l'ordre a l'unite courante de se deplacer vers les coordonnees indiquees */
#define DEPLACER_VERS(xCible, yCible) {\
                                PP_Pos p;\
                                p.x = xCible;\
                                p.y = yCible;\
                                PP_Unit_ActionOnPosition (current_unit, MOVE, p);\
                              }
#endif
```

**AUTHOR :** Mathieu MURATET
**TITLE :** Conception, Implementation and Evaluation of a Real Time Strategy Serious Game Dedicated to Introduce Programming Fundamentals
**PhD SUPERVISOR :** Pr. Jean-Pierre JESSEL
**DATE OF VIVA :** 2nd December 2008

---

**ABSTRACT :**
Video games are as much a part of students' culture as TV, movies or books. Recently, students are becoming less and less interested in science. Research on computer science teaching deals with theses problems in order to find a solution to attract and retain students in computer science. A promising solution consists in using students' culture on video games in order to motivate them to invest time in programming.
In this context, this document presents the conception, the implementation and the evaluation of a serious game dedicated to introduce programming fundamentals. This game is based on a real time strategy game where programming is a way to interact with the game. Thanks to the Prog&Play system, the serious game has been evaluated in several teaching contexts.

---